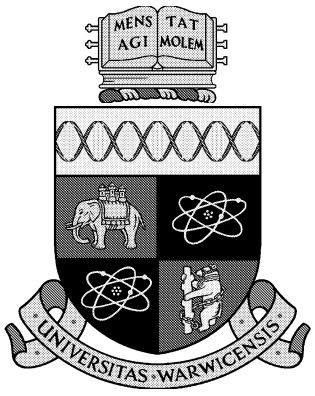


Neural Network Image Reconstruction for Nondestructive Testing

Andrew Charles Pardoe B.Eng.

A thesis submitted in satisfaction of the requirements for the degree of Doctor of Philosophy in Engineering

University of Warwick

Department of Engineering

November 1996

Contents

Declaration

I hereby declare that this thesis is my original work. To the best of my knowledge no part of this thesis has been submitted for the award of a degree other than the present submission. Some of the results within this thesis have been previously published in conference papers and journals.

Summary

Conventional image reconstruction of advanced composite materials using ultrasound tomography is computationally expensive, slow and unreliable. A neural network system is proposed which would permit the inspection of large composite structures, increasingly important for the aerospace industry. It uses a tomographic arrangement, whereby a number of ultrasonic transducers are positioned along the edges of a square, referred to as the sensor array. Two configurations of the sensor array are utilized. The first contains 16 transducers, 4 of which act as receivers of ultrasound, and the second contains 40 transducers, 8 of which act as receivers. The sensor array has required the development of instrumentation to generate and receive ultrasonic signals, multiplex the transmitting transducers and to store the numerous waveforms generated for each tomographic scan. The first implementation of the instrumentation required manual operation, however, to increase the amount of data available, the second implementation was automated.

Preliminary experiments were performed to generate 4 by 4 pixel output images for the location of defects. Investigation of the various pre-processing techniques used on the received ultrasonic waveforms indicated that frequency domain techniques gave optimum results. This was followed by an extensive investigation of the neural network's ability to determine defect size in addition to defect location. Separating the tasks of defect sizing and locating was found to give the most reliable results with the location task being limited to similar defect sizes to that used to train the neural network. An investigation of the maximum resolution possible with the given experimental system followed, and led to a system with a resolution of up to 16 by 16 pixels (each pixel was 5mm by 5mm), however, images were only possible if the validation material had similar anisotropic properties to that used to train the neural networks.

Two additional applications of neural networks to non-destructive testing have been investigated. The first allowed the enhancement of conventional ultrasonic C-scan images. Initially, experiments were performed with line-scan data and demonstrated that a neural network could classify defect size. A neural network which sampled sub-images of an original C-scan image and produced enhanced sub-images was developed and lead to a system which combined a number of enhanced sub-images to form a single enhanced image that represented the actual physical size of the defect better than the original C-scan image. In addition, the system demonstrated that it was capable of performing general image enhancement and could potentially implement several image enhancement techniques with one neural network. The second application utilized voltage tomography for the detection of localized defects. Initially defect sizing and locating was separated with different neural networks trained and tested with simulated data and validated with experimental data. Finally, a single neural network was trained to output both location and depth in the form of an image.

Publications

Pardoe, A. C., Hutchins, D. A., Mottram, J. T., and Hines, E. L. Neural networks applied to ultrasonic tomographic image reconstruction. *Neural Computing & Applications*, 5(2), April 1997.

Pardoe, A. C., Hutchins, D. A., Mottram, J. T., and Hines, E. L. Image reconstruction of composite materials using neural networks. *Invited Paper at IOP Meeting on Artificial Intelligence for Materials Processing and Characterisation*, November 1995.

Pardoe, A. C., Hutchins, D. A., Mottram, J. T., and Hines, E. L. High resolution image reconstruction of polymer composite materials using neural networks. *Review of Progress in Quantitative Nondestructive Evaluation*, 15:853–860, July 1995.

Pardoe, A. C., Hutchins, D. A., Mottram, J. T., and Hines, E. L. Applying neural networks to ultrasonic tomographic inspection of composite materials. In Orchard, G., editor, *Proc. of Third Irish Neural Networks Conference*, pages 243–251. INNA, Belfast, N. Ireland, September 1993.

Hutchins, D. A., Mottram, J. T., Hines, E. L., and Pardoe, A. C. Use of neural networks for ultrasonic tomography inspection of polymer composites. In *Ultrasonics International 93*, pages 779–782. Butterworth-Heinemann, 1993.

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Back Propagation
CFM	Continuous Filament Mat
CFRC	Carbon Fibre Reinforced Composite
DSP	Digital Signal Processing
FalAR	False Alarm Rate
FFT	Fast Fourier Transform
FN	False Negative
FP	False Positive
FSM	Field Signature Method
GFRC	Glass Fibre Reinforced Composite
GPIB	General Purpose Interface Bus
MLP	Multi-layer Perceptron
NDT	Non-destructive Testing
NN	Neural Network
PC	Principal Component
PCA	Principal Component Analysis
PosPV	Positive Predictive Value
PZT	lead zirconate titanate
QP	Quickprop
RBF	Radial Basis Function
Sens	Sensitivity

SMSE	Sum of Mean Squared Error
SOM	Self Organising Map
Spec	Specificity
TN	True Negative
TOF	Time Of Flight
TP	True Positive

Symbols

A	amplitude (m)
a	index to array pin pair
a_0	first antisymmetric Lamb mode
a_1	second antisymmetric Lamb mode
a_2	third antisymmetric Lamb mode
B	number of nearest neighbour cluster centres
b	reference to one of the B nearest cluster centres
c	index to a classification (defect location)
c	index to winning neuron in a competitive layer
d	index to a different classification from c
D	composite plate thickness (mm)
δ_{o_k}	delta
ε	error
η	learning rate
η^t	learning rate at time interval t
η_o	initial learning rate
η_{mod}	modified learning rate
f	ultrasonic frequency (Hz)
$f()$	activation function
$\text{Fc}_{V_a^t}$	fingerprint coefficient given the pin pair voltage V_a^t
g	fractional reduction in the learning rate to give η_{mod}
i	index to input neuron

I	total number of input neurons
$\mathbf{I}$	identity matrix
$_j$	index to hidden neuron
J	total number of hidden neurons
$_k$	index to output neuron
K	total number of output neurons
λ	eigenvalues of $\mathbf{\Sigma}$
λ_l	wavelength of a longitudinal (compression) wave (m)
λ_t	wavelength of a transverse (shear) wave (m)
μ_o	initial neighbourhood function
μ^t	neighbourhood function at time interval t
N_p	number of patterns
N_w	number of weights
N_k	number of output neurons
N_h	number of hidden neurons
N_i	number of inputs
N_{e_c}	total number of examples of target classification c
N_{mc}	maximum number of mis-classifications within a data set
o_k	output from neuron k
$\hat{o}$	estimate of o
ψ	parameters of linear regression
ϕ_i	eigenvector of $\mathbf{\Sigma}$
p	reference to pattern
P	total number of training patterns
ρ_j	receptive field of neuron j
$\mathbf{\Sigma}$	covariance matrix
σ	momentum
S_k	summation of weighted inputs for the output neuron k
S_j	summation of weighted inputs for the hidden neuron j
s_0	first symmetric Lamb mode

s_1	second symmetric Lamb mode
s_2	third symmetric Lamb mode
t	time interval
T	total number of training time intervals
t_i	target for output neuron k
$\mathbf{t}$	target vector for one input pattern
$\mathbf{T}$	target matrix containing all the target classifications
v_{ji}^t	weight between input neuron i and hidden neuron j
v_l	velocity of a longitudinal wave (ms^{-1})
v_t	velocity of a transverse wave (ms^{-1})
V_a^t	voltage of sensor array pin pair a at time t (ms^{-1})
V_r^t	voltage of reference pin pair at time t (ms^{-1})
w_{kj}^t	weight between hidden neuron j and output neuron k
y_j	output from hidden neuron j
z_i	input to neuron i
$\mathbf{z}$	input vector of one pattern
$\mathbf{Z}$	input matrix containing all the input patterns

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Nondestructive Testing

It is important that the properties of engineering components and structures be determined either during manufacture or in service. This is best achieved without damage to the component and its material(s), and the process of doing this is known as nondestructive testing (NDT). This is also known as nondestructive evaluation, or NDE. Typically, NDT methods are designed to look for flaws or defects that have either been introduced during manufacture, or which have been caused by the subsequent use of the component in an application. NDT is becoming increasingly used in many safety-critical areas, including the aerospace and nuclear sectors.

In this thesis ultrasonic measurements will be of particular interest [1], although potential drop methods are also used [2]. There are, in fact, many different methods used in NDT for a wide range of materials. Examples include thermography, radiography and magnetic particle inspection [3–8]. Despite the wide range of techniques available, ultrasonic testing remains the most widely used method for flaw detection. This is because it is relatively cheap and easy to use. In its conventional form, a single transducer is used to send an ultrasonic signal through a sample. This then reflects from various features within the sample, and a signal is scattered back to the transducer, which can act as both a source and receiver. This is known as pulse-echo testing. If a separate source and receiver are used, then pitch-catch operation results.

The bulk of this thesis describes an instrumentation system, which uses an array of miniature transducers to detect defects in advanced fibre reinforced polymer composite materials. Neural networks are used to form images, the aim being to produce a potentially rapid imaging method for the detection of defects, such as delamination cracks, over large areas.

This is becoming increasingly important in applications such as aircraft structures, where laminated composites are increasingly being used. In this application, damage such as delamination between layers of fibre reinforcement can result from impact (e.g. a bird strike), or faulty manufacture, which can, in a catastrophic fashion, affect the strength of a component (such as an aircraft wing) often without visible signs of damage from the surface. A review of aircraft wing design [9] has indicated the tolerable levels of damage to various areas of a wing and concludes:

> "Fairly large defect sizes, e.g. 25mm diameter, may be acceptable without repair in certain areas ... such defects would not be expected to grow during the life of the aircraft and can be left (unrepaired) and simply subjected to periodic inspection"

Thus, it is important to be able to detect and monitor such defects in service, and the present work describes progress towards such a possibility.

In the following four sections, background information is presented before being further developed in subsequent chapters. Ultrasonic waves are described briefly, before a description of the typical structure of fibre reinforced composites. A review is then presented of previous work, in which neural networks have been applied to NDT methods in general, and ultrasonic measurements in particular. This is followed by a description of the NDT technique, known as ultrasound tomography, which is used throughout this work.

1.2 Ultrasound

Unlike electromagnetic waves, sound waves are unable to travel through a vacuum. The existence of a sound wave requires the presence of particles of matter, in the form of solid, liquid or gas, which may vibrate. Ultrasound has a frequency range above that of the human ear, with typical frequencies used for NDT, ranging up to 20MHz [4].

In an infinite homogeneous isotropic material there can only exist two ultrasonic modes, namely, longitudinal (often referred to as compression) and transverse (also known as shear) waves, which are collectively called bulk waves. The velocity of these two modes will depend on the stiffness of the material, (the greater the stiffness the higher the velocity for a given frequency), and may be expressed in terms of the materials Young's modulus, Poisson's ratio and density [1, 3]. For a finite homogeneous isotropic material having one free surface an additional mode exists, called a Rayleigh or surface wave and, as the name suggests, it travels along the surface of the material. Rayleigh waves may exist on both surfaces of a finite material if its thickness is significantly larger than the wavelength of ultrasound. For homogeneous isotropic plate materials, where the thickness is approaching the ultrasonic wavelength, the Rayleigh mode will be increasingly superseded by Lamb modes (often known as plate or guided modes). These Lamb modes generically result from the superposition of a bulk wave undergoing multiple reflections and mode conversions between plate surfaces. Lamb waves may be considered as a series of modes of different order, with each being grouped into one of two types according to the displacement about the plate's mid-plane, namely, symmetric and anti-symmetric [10].

Anisotropic materials, such as advanced fibre reinforced polymer composites, have elastic properties which vary according to propagation direction. Thus the velocity of the bulk, surface and plate waves in these materials will depend on the propagation direction.

For materials with either micro- or macro-structural inhomogeneities, for example, grain boundary discontinuities in metals, fibre-matrix boundaries or inter-laminar interfaces in composites, there are a number of ultrasonic phenomena occurring, namely, reflection, refraction, mode conversion and attenuation. These phenomena have the

effect of significantly complicating conventional NDT techniques in the detection and location of defects.

Attenuation or energy loss within an ultrasonic wave propagating through a material may be accounted for by several physical mechanisms, such as, heat conduction, viscous friction, elastic hysteresis and scattering. These mechanisms are frequency dependent and difficult to determine theoretically. Scattering which involves an ultrasonic wave being reflected and refracted due to inhomogeneities within a material may become prominent when a defect or flaw is present, especially if the size of the defect is larger than the ultrasonic wavelength. Thus, a defect within a material may be considered to act as an inhomogeneity with respect to its effect upon ultrasound, and detection of the reflections caused by scattering forms the basis of many conventional ultrasonic NDT techniques.

There are a number of methods used to generate and detect ultrasound. By far the most common method, used for the majority of this work, is the use of piezoelectric transducers [11,12] which may be used for generation and detection of ultrasound. Another popular method of ultrasound generation is the pulsed laser. Detection methods also include electromagnetic and electrostatic acoustic transducers (often referred to as EMATs and ESATs), knife edge detectors and interferometers. Each detection method has different sensitivity and bandwidth performance. Typically, the piezoelectric transducers have the greatest sensitivity, but with a reduced bandwidth.

1.3 Fibre Reinforced Polymer Composites

A composite material is defined by Schwartz [13] as follows: "A composite material is a material brought about by combining materials differing in composition or form on a macroscale for the purpose of obtaining specific characteristics and properties." Typically, composites combine either particles, fibres or flakes within a matrix, and may be layered to form a laminated composite.

Because a composite material is made from combining several different materials, the resulting physical properties of the composite are a combination of the properties of the component materials. Thus, by altering the internal structure, a given physical property of the composite material can be dominated by one of the component mate-

rials. Indeed, this is the main advantage of composites, with most modern composite materials being designed to have very specific physical properties. This is increasingly important for the aerospace sector where components are required to have high strengths and stiffnesses but with a minimum weight.

All of the composite materials used in this study are known as fibre reinforced polymers. There are several fibre types which are commonly used for such composite materials, including glass, carbon, aramid, boron and silicon carbide. In this work only glass and carbon fibres were used. The glass fibre composites were manufactured using the pultrusion process [14]. Shown in Figure 1.1, the process combines the different layers of the composite material with the resin matrix in stages before being pulled through a preformer to give the material the desired shape. Finally, the material enters a forming and curing die where under controlled heating conditions the resin matrix sets to produce the hardened composite material [15]. This process is typically used to produce continuous lengths of sections with constant cross-sectional dimensions. There are a wide range of shapes that can be pultruded and a number of large sections are often used in civil engineering applications. The fibre layers of the material are alternated between a mat of randomly placed continuous fibre and uni-directional fibre bundles (rovings). The rovings are aligned with the pultrusion process to produce sections with the highest mechanical properties in the longitudinal direction. The polymer (resin) matrix used was an isophthalic polyester combined with a calcium carbonate filler (filler may also affect the ultrasonic propagation). The carbon fibre composite used in this work was of the pre-preg type and was manufactured using a pressure moulding process [16]. The fibre layers are uni-directional with all layers having the same orientation, hence the material is uni-directional. The matrix used was an epoxy polymer without a filler.

Because of the nature of composite materials, their mechanical properties are at best pseudo-isotropic and typically anisotropic. Although conventional ultrasonic NDT techniques have been used successfully to inspect composite materials, both generally [17,18] and during the manufacture process [19,20], the ability of conventional ultrasonic NDT techniques to produce reliable and relevant images of defects is significantly reduced by materials which exhibit anisotropic behaviour.

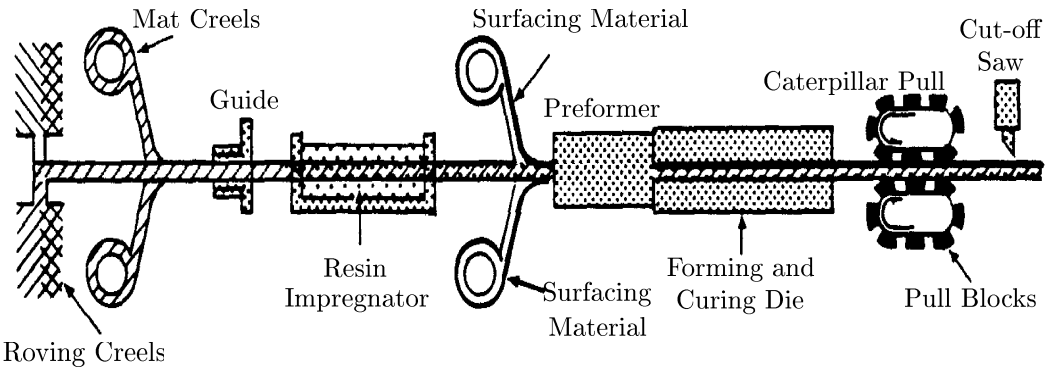


Figure 1.1: The pultrusion process for composite materials (Courtesy of Morrison Molded Fiber Glass (MMFG) company, Bristol, Virginia, USA).

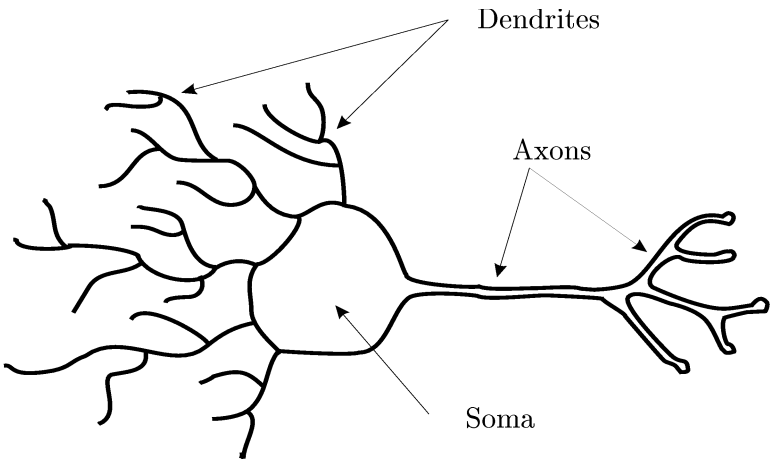


Figure 1.2: The three main components of a biological neuron.

1.4 Neural Networks and Applications in Ultrasonic NDT

Originally the inspiration for the field of study known as Artificial Neural Networks (ANNs), commonly referred to as Neural Networks (NNs), came from the study, conducted by neuroscientists and psychologists, of biological neurons within the human brain. The three main components of a biological neuron, shown in Figure 1.2, are the Axons which transmit the neuron's state or activity, the Dendrites which connect to either external stimuli or to the axons of other neurons and collect the neuron's input stimuli, and the cell body or Soma, which processes the incoming stimuli (from the dendrites) to give the neuron activity.

The human brain contains billions of neurons and as suggested above they are all inter-connected to form a network of neurons (approx. 10^{14} connections), with the axons of one layer connected to the dendrites of the next. However, current artificial neural network technology, even with the most powerful computers, cannot approach such a huge capacity [21]. An artificial neural network from the engineering point of view, is not a model of biological neurons, but is simply a parallel processing technique inspired by both the parallel nature and simplicity of computation of biological neurons.

By far the most commonly used neural network technique for engineering applications is that of back-propagation (BP) [22–24]. It structures neurons into layers, typically three layers which are referred to as the input, hidden and output layers. In general terms the BP neural network can be considered to be able to map any input data onto any output data [25,26]. The neural network undergoes a learning or training process whereby input data (examples of the input) are presented to the neural network along with the desirable output data (examples of the output). After a neural network has been successfully trained it may be used to classify previously unseen input data. This is done by presenting the trained neural network with the unseen input data, the neural network will then produce outputs that it believes classifies the presented input.

Neural networks may be used not only for classification problems such as credit screening and fraud detection [27], speech [28] and hand-writing [29] recognition, but also for regression problems, example applications of which include stock market and weather prediction. For both forms of problem the main advantage of neural networks

is that specification of the underlying physical model for the given application is not required. Typically neural networks will be used with applications were the physical model is extremely complicated or even unknown. The application area of interest here is that of nondestructive testing, and neural networks have been applied to several related fields, including medical imaging and diagnosis [24,30] and manufacturing quality control [31–33].

Due to the large amounts of data generated from ultrasonic NDT, automated methods of data analysis have always been sought. Artificial intelligence techniques have been frequently used to analyse ultrasonic data. Initially, expert systems were used [34–36]. However, McNab and Dunlop [37] gave a recent review of data analysis techniques used for ultrasonic NDT and illustrated that neural networks have also been successfully used. Windsor [38] concludes that a neural network approach, rather than an expert system, for the analysis of ultrasonic data produces greater confidence. This is because the neural network can determine if it is unable to classify a given set of ultrasonic data.

Neural networks have been successfully used for the classification of ultrasonic signals, for the evaluation and location of defects, by many researchers, including Udpa *et al* [39,40], Damarla *et al* [41], Challis and Bork [42,43], Sachse *et al* [44] and Bull and Smith [45]. The interpretation of ultrasonic signals by a neural network, specifically for the evaluation and location of defects in particular materials, has been investigated by Shahani *et al* [46] and Baker and Windsor [47], for defects in metals, and by Damarla *et al* [41] and Thomsen and Lund [48] for defects in isotropic composites. This work will apply neural networks to ultrasonic data taken from anisotropic composite materials using a tomographic method of data collection.

1.5 Tomographic Imaging

Tomographic imaging is a technique which determines a spatially varying parameter of an object's cross-section from signals which have travelled through the object [49, 50]. Typically the object is surrounded by a number of sensors which transmit and receive signals along paths (individually called raypaths and collectively referred to as

projections) through the object.

Ultrasonic tomography has been used in the past for image reconstruction [51–53]. Its main application area has been in medical ultrasound [54–56], but even here there have not been many applications, because of complications due to multiple reflections, refractions, etc. Because of material anisotropy the application of traditional tomography for fibre reinforced composites is also complicated. Although work has been done in applying conventional tomography to such anisotropic materials, such as rocks in seismic tomography [57], it is a complicated method and has various degrees of success [58, 59].

For the above reasons, a neural network method of image reconstruction was considered a potential benefit to image reconstruction in the presence of complications where conventional tomography would have difficulties.

The tomographic geometry used for the majority of the work presented in this thesis is shown in Figure 1.3. It allows inspection of large areas with minimum risk of missing small areas and permits the scan area to be subdivided into square spatial elements for neural network training.

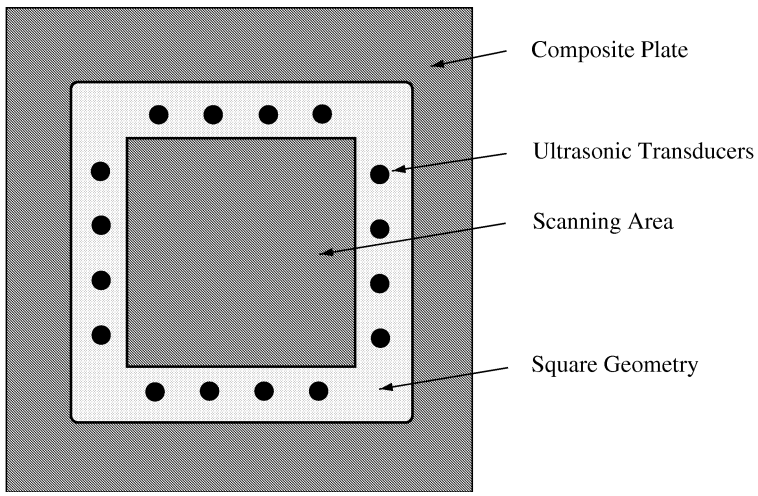


Figure 1.3: Top view of basic geometry.

1.6 Thesis Overview

The four previous sections have outlined the appropriate background material necessary to understand the main aim of this work. The work presented in this thesis combines the four fields of study of ultrasonic NDT, composite materials, neural networks and tomography. The main objective has been to demonstrate that a neural network system can be devised to perform image reconstruction of composite materials using ultrasound tomography. Conrath [60] has previously investigated the application of neural networks to ultrasonic tomographic reconstruction for the classification of defect size (the neural network had a single output representing the defect size) and concluded that a neural network is able to learn the ultrasonic projection data. He did however need to emphasize the problem that, even for a moderate pixel resolution image, the required neural network would be extremely large and difficult to train.

In Chapter 2, an introduction to neural network theory is given with a detailed description of the different neural network learning algorithms used throughout this work. In addition to the theoretical details, the main practical issues involved with the development of a neural network system are discussed together with relevant guidelines used to aid in the search for the optimum system. This is followed by giving appropriate neural network performance metrics necessary for comparisons.

Chapter 3 describes the experimental apparatus designed to collect ultrasonic tomography data. The various stages of development of the apparatus are detailed and the two stages used for data collection are illustrated. The software, written by the author, to control the automated data collection is also detailed. Descriptions of the different composite materials used for experimentation are given along with an investigation of the materials' ultrasonic behaviour.

Before a neural network can be used, training and testing data are required. Chapter 4 details the methods employed for data acquisition and pre-processing. The different types of defect used together with their location within the image area are described. Simulation software, written by the author, has been developed to determine the optimal arrangement of the receivers within the tomographic geometry. Next an investigation of the ultrasonic modes present in the experimentally generated waveforms

allow the identification of a pre-processing technique suitable to reduce the dimensionality of the collected tomographic data. Various pre-processing techniques are then compared using cluster analysis to determine the technique which contains the most information relating to defect location.

Chapter 5 begins with regression analysis of the training data sets and illustrates the associated problem, referred to as image biasing. An extensive investigation of the neural network performance with the initial data sets is given and leads to a strategy for further data collection. This is followed by a description of novel methods to reduce the effects of image biasing which are most significant with higher resolutions. This gives rise to the development of a final system which is evaluated with validation data.

A neural network approach to improve the performance of conventional ultrasonic NDT is given in Chapter 6. Neural networks are initially applied to simple ultrasonic line-scans in order to illustrate their ability to correctly classify defect size. Then a novel neural network system is developed which enhances original C-scan images to produce an image which better reflects the actual defect size.

The application of neural networks to voltage tomography is explored in Chapter 7 for the detection of localized corrosion within metal pipework. Neural networks have been trained with simulated data to determine both the size and location of localized defects. Then experimental data is used to validate the neural network performance.

Chapter 2

Artificial Neural Networks

2.1 Introduction

An artificial neural network has the ability to learn an input-output function without *a priori* knowledge of the relationship between them. This learning (or training) phase is achieved by presenting the neural network with examples of the input-output function (known as training data). Thus, a properly trained neural network, which has been properly tested in terms of its generalisation ability using testing data, can be used to generalise for "unseen" data (referred to as validation data). Typically the neural network requires a large number of varied examples to learn, which should adequately cover the problem space.

This chapter is divided into four parts. The first, detailed in Section 2.2, presents a general definition of an artificial neural network. The next part, given in Section 2.3 to Section 2.5, details the four different neural network learning algorithms which will be used throughout this thesis. The third part, Section 2.6 to Section 2.8, describes the practical issues involved with training a neural network and optimizing a neural network system. The final part, presented in Section 2.9, describes the performance measures used to compare trained neural network performance.

2.2 Defining an Artificial Neural Network

Because of the many disciplines studying neural networks, there is no generally accepted formal definition of a neural network. However, for the purpose of this study, the following is given, by the author, as a general definition:

> A neural network is composed of three main components, the neuron model, the connectionist model and the learning paradigm. The neuron model defines the functional mapping from inputs to output of all the neurons in the ANN. The connectionist model defines the architecture, which includes both the arrangement and connectivity of the neurons within the ANN. Finally, the learning paradigm specifies the algorithm used to modify the values associated with the connections between neurons.

2.2.1 The Three Components of an Artificial Neural Network

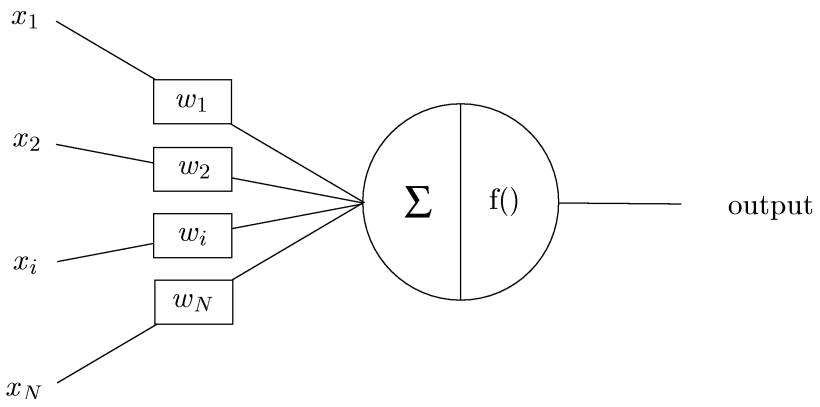
As defined above, an artificial neural network has three main components; the neuron model, the connectionist model and the learning paradigm.

2.2.1.1 The Neuron Model

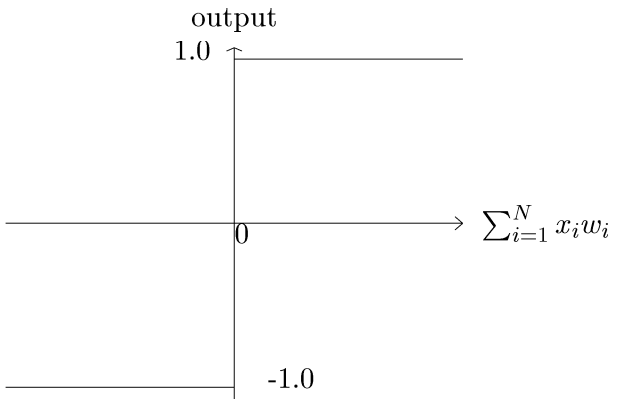
The neuron model defines the relationship between the neuron's inputs and connections (known as weights in this model) and the neuron's output. As with the axons and dendrites of the biological neurons which vary their connection strengths, the weights within this model are also adjustable. Initial work by McCulloch and Pitts [61] lead to the development of the Perceptron by Rosenblatt [62] which now forms the basis for most current neuron models. A perceptron is shown in Figure 2.1(a) and expressed mathematically as,

$$\text{output} = f\left(\sum_{i=1}^{N} x_i w_i\right), \tag{2.1}$$

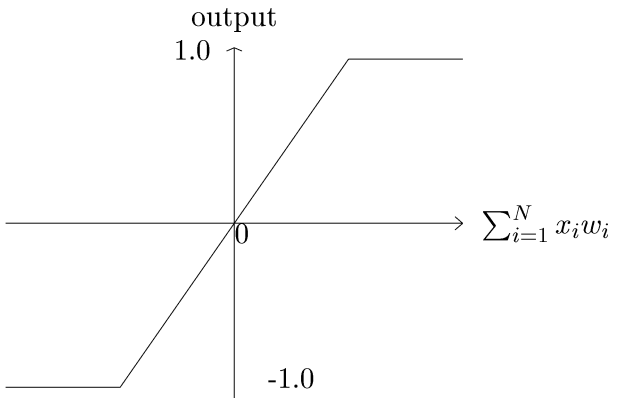
where $f()$ is known as the activation function, w_i is the weight of connection i, x_i is the input to connection i and N is the number of inputs feeding into the neuron. Typical traditional activation functions are given in Figure 2.1(b) and Figure 2.1(c). For a



(a) Schematic of a Perceptron.



(b) Signum activation function.



(c) Linear activation function.

Figure 2.1: The Perceptron.

given ANN the neuron model will typically be the same for all neurons in the same layer of the connectionist model (either a linear or non-linear activation function).

2.2.1.2 The Connectionist Model

The connectionist model describes a distributed architecture of computational processors which are sometimes called nodes or processing elements, but more often referred to as neurons. Typically, the neurons are arranged into layers with connections between consecutive layers. The connections are actually the weights of the neuron model. It is the combination of these layers of neurons and inter-layer connections which constitutes the connectionist model. A connectionist model which forms the basis for many modern day neural networks was devised by Rosenblatt [63] specifically for the perceptron, and is called the multi-layer perceptron (MLP). Figure 2.2 illustrates the basic MLP structure which is fully connected, meaning that each and every neuron in a given layer is connected to all neurons in adjacent layers. It is assumed that a connectionist model is fully connected unless specified otherwise.

There appears to be no standard naming convention for a given connectionist model. For example, the model shown in Figure 2.2 seems to have three layers of neurons and thus would be referred to as a "three layer network". However, the input layer of neurons do not act like the neurons described in Section 2.2.1.1 but simply distribute the input data to the inputs of the hidden layer of neurons. It is the authors view that such a connectionist model should be referred to as a "two layer network", thus referring to the number of weight layers (or actual 'processing' neuron layers).

2.2.1.3 The Learning Paradigm

The learning paradigm determines the functionality of the neural network by specifying the way in which the adjustable weights of the connectionist model are modified during the learning or training phase. An artificial neural network learns by example, and so the learning phase involves the presentation of a training data set to the neural network. Often referred to as input data, this training data set contains many data examples, sometimes known as input patterns, of one or more input variables.

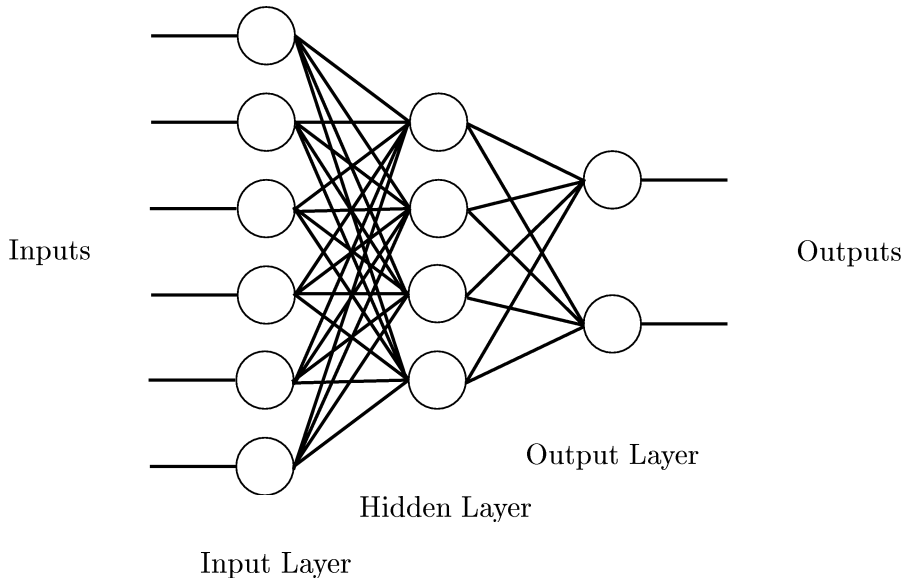


Figure 2.2: Basic structure of an MLP connectionist model.

The learning paradigm implements an algorithm which adjusts the weights according to a relationship with the input patterns. Typically this is an iterative process, whereby the training data set is presented to the neural network many times. There are two main learning paradigm categories, known as supervised and unsupervised learning algorithms.

Supervised learning algorithms require that the training data consists of both an input data set and target classifications, whereby each pattern within the input data has an associated target classification which defines the desired output of the neural network. Here, the task is to associate each input pattern with its target output. This is typically achieved by minimizing a global error (usually the difference between the target output and the actual neural network output over all the training examples) by gradient descent. Figure 2.3 illustrates the plot of global error against weight space, where weight space represents all the possible combinations of values for all the weights within the neural network. This demonstrates that by changing the weight values in a direction specified by the error gradient (from 1 to 2 and then 2 to 3) will lead to a reduction in the global error until a minimum is approached.

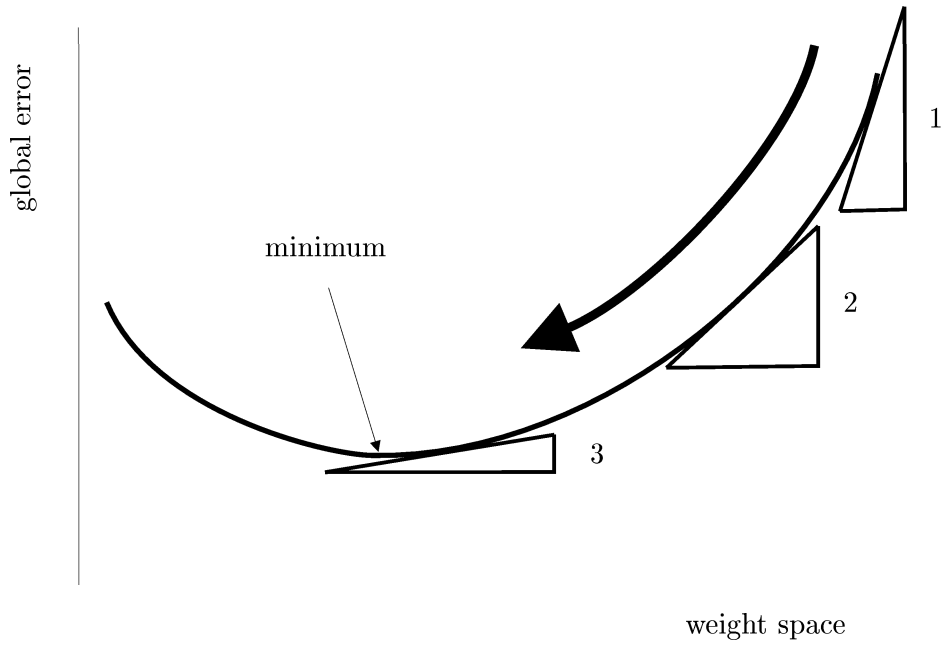


Figure 2.3: Error gradient descent.

Unsupervised learning algorithms attempt to find inherent information within the input data (does not require user defined target classifications). They are often used to extract clusters of similar input patterns in the data set [64].

2.2.2 Learning Algorithms

The neuron model, connectionist model and the learning paradigm are usually combined to form the 'Learning Algorithm'. However, typically an artificial neural network will be specified by the learning paradigm alone as the connectionist and neuron model are inferred.

Four different learning algorithms have been used throughout this work. The majority of the work has concentrated on the back-propagation algorithm and its modified variant, quickprop. Radial basis function networks habe also been used. Initial analysis of the ultrasonic data will be performed using the self organising map.

2.3 The Back-Propagation Algorithm

The back-propagation (BP) learning algorithm employs a supervised learning paradigm, and has a similar connectionist model to that shown in Figure 2.2. Specifically, it is based on the multi-layer perceptron architecture devised by Rosenblatt [63], and devel-

oped by several researchers including Rumelhart [65]. One addition to this architecture is that of a bias neuron, which is connected to all neurons in the hidden and output layers. The bias neuron has a single input which is fixed at a value of 1. This modification to the standard MLP architecture was introduced by Dayhoff [21] to improve the convergence properties of the NN, by supplying a constant term to the weighted sum at the hidden and output neurons.

The BP algorithm uses the delta learning rule [66], which is occasionally referred to as the Widrow-Hoff rule [67,68] and is based on the Hebbian learning rule [69]. The delta rule uses a gradient descent method to adjust the neural network's weights in order to minimise a global error function.

The following derivation of the BP algorithm is based on that given by Zurada [70], and assumes a connectionist model with I inputs, J hidden neurons and K output neurons, as shown in Figure 2.4, where $\mathbf{v}= \{v_{11}, \ldots, v_{JI}\}$ is the hidden layer of weights and $\mathbf{w}= \{w_{11}, \ldots, w_{KJ}\}$ refers to the output layer of weights. The input vector to the neural network, $\mathbf{z}$, is taken from the input data set $\mathbf{Z}= \left\{\mathbf{z}^1, \ldots, \mathbf{z}^p, \ldots, \mathbf{z}^P\right\}$ which contains P patterns with each pattern, $\mathbf{z}^p$, having I variables, $\mathbf{z}^p = \left\{z_1^p, \ldots, z_I^p\right\}$, such that $\mathbf{z}= \mathbf{z}^p$ where $1 \leq p \leq P$. For a given input pattern $\mathbf{z}$, the forward pass of the ANN is calculated, as defined by

$$o_k = f\left(\sum_{j=1}^{J} y_j w_{kj}\right) \quad \text{with} \quad y_j = f\left(\sum_{i=1}^{I} z_i v_{ji}\right), \tag{2.2}$$

where $\mathbf{y} = \{y_1, \ldots, y_J\}$ is the output of the hidden layer of neurons, $\mathbf{o} = \{o_1, \ldots, o_K\}$ is the neural network output and $f()$ is the activation function.

Next the global error function is given by

$$E_p = \frac{1}{2}\sum_{k=1}^{K}\left(t_k - o_k\right)^2. \tag{2.3}$$

It combines the differences between the neural network's outputs, as calculated by equation 2.2, and the desired output targets, $\mathbf{t}$, for the given input pattern. Where the target classifications for the input data set, is given by $\mathbf{T}= \left\{\mathbf{t}^1, \ldots, \mathbf{t}^p, \ldots, \mathbf{t}^P\right\}$ with each $\mathbf{t}^p$ having K variables, $\mathbf{t}^p = \left\{t_1^p, \ldots, t_K^p\right\}$, thus $\mathbf{t}= \mathbf{t}^p$.

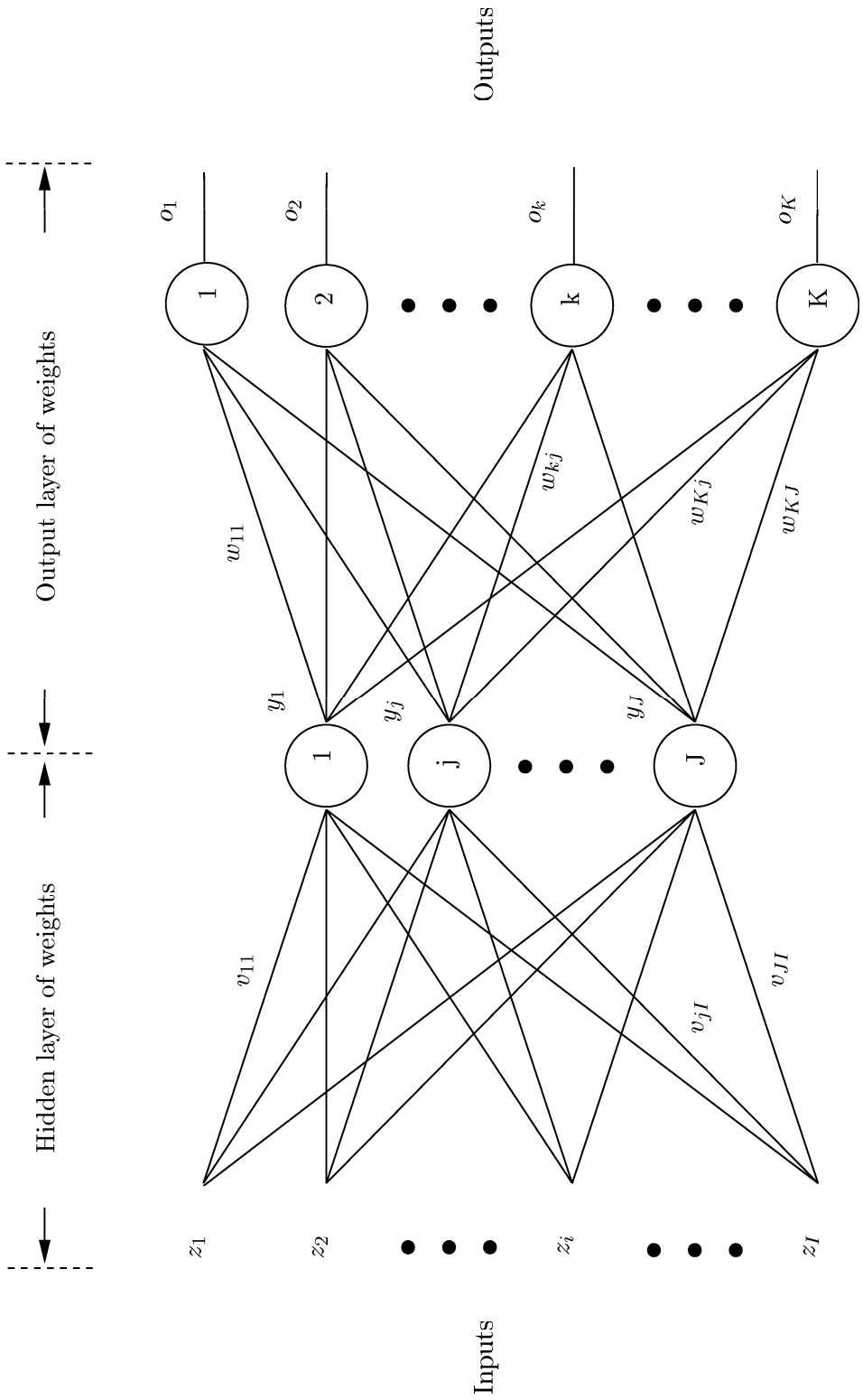


Figure 2.4: The connectionist model for the BP algorithm.

Note that this global error is only for the presentation of one input pattern p. The presentation of one pattern to the ANN is called a cycle while the presentation of all P patterns to the ANN is called an epoch. There exist two strategies for the updating of weights during an epoch. The first is called continuous updating, and modifies the weights after the presentation of each pattern within the training data set. The second, called periodic updating, sums the global error for all patterns within the training data set before modifying the weights. A more detailed account of these two strategies will be given later in Section 2.7.4, but for the purpose of this derivation we shall assume that either strategy is to be used; thus, further reference to the global error function will be without the subscript p.

The delta rule adjusts the output layer weights within a network according to the following equation:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} \tag{2.4}$$

where η is known as the learning rate, w_{kj} is the weight value between output neuron k and hidden neuron j, Δw_{kj} is the change in weight w_{kj} and E is the global error. If we define δ_{o_k} to be the error signal term for output neuron k as

$$\delta_{o_k} \triangleq -\frac{\partial E}{\partial (S_k)} \quad \text{where} \quad S_k = \sum_{j=1}^{J} w_{kj} y_j \tag{2.5}$$

then applying the chain rule to the partial derivative of equation 2.4 gives

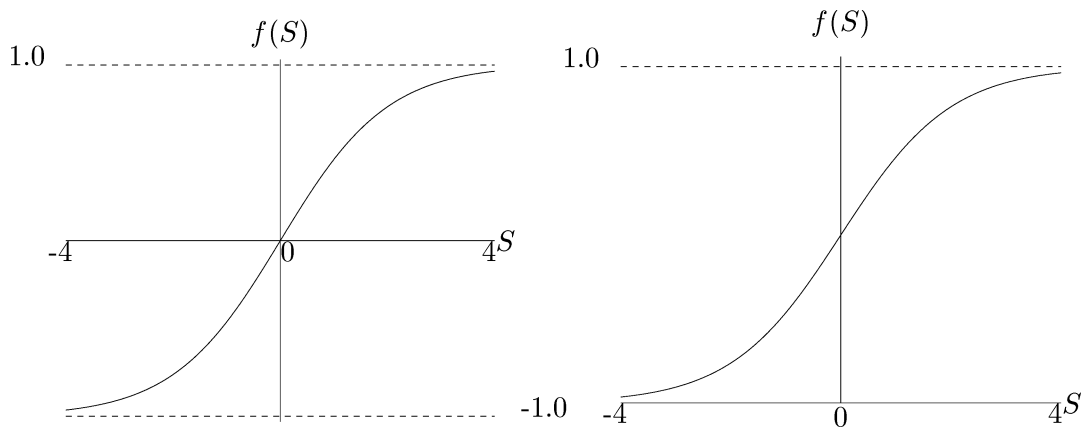
$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial (S_k)} \cdot \frac{\partial (S_k)}{\partial w_{kj}}. \tag{2.6}$$

Noting that

$$\frac{\partial (S_k)}{\partial w_{kj}} = y_j \tag{2.7}$$

and by combining equations 2.4, 2.5, 2.6 and 2.7 we obtain

$$\Delta w_{kj} = \eta \delta_{o_k} y_j. \tag{2.8}$$



(a) bipolar. (b) unipolar.

Figure 2.5: Sigmoid functions.

Given equation 2.5 we have

$$\delta_{o_k} = -\frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial (S_k)}. \tag{2.9}$$

Taking the partial derivative of equation 2.3 gives

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{2.10}$$

and noting that

$$f'(S_k) \triangleq \frac{\partial o_k}{\partial (S_k)} \tag{2.11}$$

equation 2.9 can be rewritten using equations 2.10 and 2.11 to obtain

$$\delta_{o_k} = (t_k - o_k) f'(S_k). \tag{2.12}$$

Because the delta rule uses the derivative of the activation function, an appropriate continuous and differentiable function must be selected (those activation functions shown in Figure 2.1(b) and Figure 2.1(c) are not continuous). Commonly used activation functions include the bipolar or unipolar sigmoid functions (sometimes referred to as hyperbolic and logistic functions), shown in Figure 2.5(a) and Figure 2.5(b) and expressed as follows

$$f(S_k) \triangleq \frac{2}{1+e^{-S_k}} - 1, \tag{2.13}$$

for the bipolar sigmoid function, and

$$f(S_k) \triangleq \frac{1}{1+e^{-S_k}} \tag{2.14}$$

for the unipolar sigmoid function. The activation function maps the summation of a neurons weights and inputs (S_k) to the neuron's output, the main difference between these two activation functions is the bounding limit placed on the neuron's output; with the unipolar function the output of the neuron is limited to values between 0 and 1, and with the bipolar function the limit is between -1 and 1. The two activation functions given in equations 2.13 and 2.14 have simple derivatives. Derived in appendix A they are expressed as

$$f'(S_k) = \frac{1}{2}\left(1 - o_k^2\right) \tag{2.15}$$

for the bipolar activation function, and

$$f'(S_k) = o_k(1 - o_k) \tag{2.16}$$

for the unipolar function.

Substituting equation 2.16 into equation 2.12 allows the expansion of equation 2.8 into the delta rule output layer weight update equation, at update interval $t+1$:

$$\Delta w_{kj}^{t+1} = \eta(t_k - o_k)\, o_k(1 - o_k)\, y_j$$

$$w_{kj}^{t+1} = w_{kj}^{t} + \Delta w_{kj}^{t+1}. \tag{2.17}$$

The application of the generalized delta rule allows the weights of the hidden layer to be updated in a similar fashion. Again using the negative gradient descent, the change in the hidden layer weights is given as

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}}. \tag{2.18}$$

Defining the error signal term for the hidden layer neurons as

$$\delta_{y_j} \triangleq -\frac{\partial E}{\partial (S_j)} \quad \text{where} \quad S_j = \sum_{i=1}^{I} v_{ji} z_i \tag{2.19}$$

and applying the chain rule

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial (S_j)} \cdot \frac{\partial (S_j)}{\partial v_{ji}} \tag{2.20}$$

gives the weight adjustment as

$$\Delta v_{ji} = \eta \delta_{y_j} z_i. \tag{2.21}$$

Thus

$$\delta_{y_j} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial (S_j)} \tag{2.22}$$

where

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j} \left(\frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2 \right). \tag{2.23}$$

Differentiating, substituting equation 2.2 and then simplifying using equations 2.12 and 2.5 we obtain

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K} \delta_{o_k} w_{kj}. \tag{2.24}$$

Given that

$$\frac{\partial y_j}{\partial (S_j)} = f'(S_j), \tag{2.25}$$

combining equations 2.24 and 2.25 allows equation 2.22 to be rewritten as

$$\delta_{y_j} = f'(S_j) \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{2.26}$$

Hence, using equations 2.21 and 2.26, and noting that the derivative of the activation function for a hidden neuron can be expressed as a function of the output of that neuron (as with the output neurons, see equation 2.16), we can form the hidden layer weight update equation, again at update interval $t+1$:

$$\Delta v_{ji}^{t+1} = \eta y_j (1 - y_j) z_i \sum_{k=1}^{K} \delta_{o_k} w_{kj}$$

$$v_{ji}^{t+1} = v_{ji}^{t} + \Delta v_{ji}^{t+1}. \tag{2.27}$$

In practice, the presentation of the training data set and the modification of the neural network's weights would occur many times, typically until the global error has reached a specified value. The details of the stop training criteria used will be discussed later in Section 2.6.1.

Much work has been conducted by many researchers to improve the training speed (convergence speed) of the standard BP algorithm. Several basic improvements which have become proven techniques include the addition of a momentum term [66] and flat spot elimination [71].

Momentum adds a portion of the previous weight change to the present weight change. In addition to increasing the speed of convergence, the momentum term prevents the NN from getting stuck in a local minimum of the error surface within weight-space. Applying the momentum term, σ, to the weight update equations 2.17 and 2.27 we obtain

$$w_{kj}^{t+1} = w_{kj}^{t} + \Delta w_{kj}^{t+1} + \sigma \Delta w_{kj}^{t} \tag{2.28}$$

$$v_{ji}^{t+1} = v_{ji}^{t} + \Delta v_{ji}^{t+1} + \sigma \Delta v_{ji}^{t} \tag{2.29}$$

Flat Spot Elimination adds a constant value, typically 0.1, to the derivative of the activation functions, expressed by equation 2.16, thus preventing the derivative from approaching zero. Thus, equations 2.27 and 2.17 can be rewritten as

$$\Delta v_{ji}^{t+1} = \left(y_j \left(1 - y_j\right) + 0.1\right) \eta z_i \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{2.30}$$

$$\Delta w_{kj}^{t+1} = \eta \left(t_k - o_k\right) y_j \left(o_k \left(1 - o_k\right) + 0.1\right) \tag{2.31}$$

2.3.1 The QuickProp Algorithm

The quickprop (QP) variation of the standard BP algorithm was conceived by Fahlman [71]. It is a heuristic approximation of the second order derivative of the global error, based loosely on Newton's method, giving the curvature of the error function.

Fahlman [71] states that:

> "Quickprop is based on two assumptions, that the error verses weight curve for each weight can be approximated by a parabola whose arms open upwards and the change in slope of the error curve, as seen by each weight, is not affected by all the other weights that are changing at the same time."

In practice, for a given epoch, for all weights, a copy of the first-order partial derivative of the error with respect to the weight and the difference between the current and previous values of the weight are kept for the next epoch. Thus the calculation of the change in weights for the next epoch is denoted by equation 2.32:

$$\Delta w^{t+1} = \frac{\left(\frac{\partial E}{\partial w}\right)^{t+1}}{\left(\frac{\partial E}{\partial w}\right)^{t} - \left(\frac{\partial E}{\partial w}\right)^{t+1}} \Delta w^{t}. \tag{2.32}$$

In the case where the current slope is in the same direction as the previous one, but is the same size or larger in magnitude, to prevent taking an infinitely large step (or moving backwards), a "maximum growth factor" is used to limit the weight change. This allows acceleration down steep slopes, but within limits.

The quickprop algorithm will, for some applications, cause some of the weights to grow very large [71]. Therefore a technique called "weight decay" is used, which penalizes large weights. As described by Hinton [72], the error function takes another term, which is the sum of squared weights (the derivative of which corresponds to the weight decay). Thus each weight continually decays towards zero by an amount proportional to its magnitude. This technique tends to prevent the neural network from becoming a look-up table.

2.4 The Self Organising Map

The Self Organising Map (SOM) devised by Kohonen [73] is a single layer network that organises itself into a topological map from a random start point, by means of an unsupervised learning paradigm. The resulting map illustrates any natural relationships amongst the input data set. As shown by Figure 2.6, the output layer of neurons, referred to here as the competitive layer, form a rectangular array of neurons and are fully connected to the inputs.

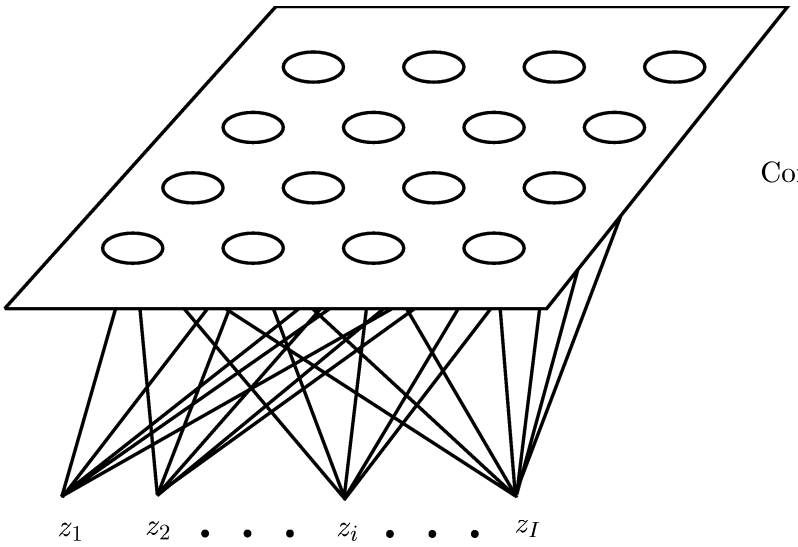


Figure 2.6: SOM connectionist model.

For a given input pattern, $\mathbf{z}$, a comparison is performed between this pattern and the weights of each of the neurons in the competitive layer, $\mathbf{w}_k$. The winning neuron, c, defined by having the minimum Euclidean distance between its weights and the input pattern, is given by

$$\|\mathbf{z} - \mathbf{w}_c\| = \min_{k=1}^{K} \{\|\mathbf{z} - \mathbf{w}_k\|\}. \tag{2.33}$$

Thus the weight update equation, at update interval $t+1$, is given by

$$\Delta w_{ki}^{t+1} = h_{ci}^{t} \left(z_i - w_{ki}^{t}\right)$$

$$w_{ki}^{t+1} = w_{ki}^{t} + \Delta w_{ki}^{t+1} \tag{2.34}$$

where h_{ci} is the neighbourhood function which determines how many neurons neighbouring the winning neuron will also have updated weights. Typically a Gaussian function is used, given by

$$h_{ci}^{t} = \frac{\eta^t}{\mu^t \sqrt{2\pi}} \exp\left(-\frac{\|r_c - r_k\|^2}{2\left(\mu^t\right)^2}\right). \tag{2.35}$$

Here, η^t is the learning rate, μ^t the width of the neighbourhood function and $\|r_c - r_k\|$

is the distance between competitive neurons c and k. Both η^t and μ^t are monotonically decreasing functions of update interval, as shown by

$$\eta^t = \eta_o \left(1 - \frac{t}{T}\right) \tag{2.36}$$

$$\mu^t = \mu_o \left(1 - \frac{t}{T}\right) \tag{2.37}$$

where η_o and μ_o are the initial learning rate and neighbourhood width and T is the total number of update intervals.

Having trained the network, each neuron in the competitive layer is associated to a class, according to a majority decision criteria based on which class from the training data set activates the neuron the most. Thus, visually the rectangular array of output neurons will form clusters of classes which stimulate adjacent neurons and thereby have inherently similar input patterns.

2.5 The Radial Basis Function

The Radial Basis Function (RBF), created by Moody and Darken [74], combines both supervised and unsupervised learning. It has a two layer connectionist model with the output neurons having a linear activation function. The hidden layer of neurons specify cluster centres within the input-space and are used to map receptive fields.

The generation of these receptive fields forms the first phase of training for the RBF network. Initially a modified k-means algorithm, similar to Kohonen learning (see Section 2.4), is used to position the cluster centres. These centres are actually represented by the hidden neurons weights.

For a given input pattern, $\mathbf{z}$, only the closest cluster centre is modified, typically the distance metric used is Euclidean although others may be used (such as City Block). The weight update equation at interval $t+1$ is given by

$$\Delta v_{ci}^{t+1} = \eta^t \left(z_i - v_{ci}^{t-1}\right)$$

$$v_{ci}^{t+1} = v_{ci}^t + \Delta v_{ci}^{t+1} \tag{2.38}$$

where η^t is the learning rate as define by equation 2.36 and c represents the hidden neuron with the closest cluster centre to the input pattern. This is repeated for all the input patterns of the training data set and until the learning rate becomes zero.

The activation function for the hidden layer of neurons is typically defined by the Gaussian

$$f(S_j) = \exp\left(-\frac{(S_j - R)^2}{\rho_j^2}\right) \quad \text{where} \quad S_j = \|\mathbf{z} - \mathbf{v}_j\|, \tag{2.39}$$

where ρ_j specifies the width of the receptive fields and is determined by a nearest neighbour heuristic and R is the Euclidean radii of $\mathbf{z}$. For a given cluster centre $\mathbf{v}_j$ let $\mathbf{v}_{j_1} \cdots \mathbf{v}_{j_B}$ be the B nearest cluster centres. So the width of each receptive field is defined as the root mean square distance of cluster centre j to the B nearest neighbour centres, as shown by

$$\rho_j = \sqrt{\frac{1}{B}\sum_{b=1}^{B} \|\mathbf{v}_j - \mathbf{v}_{j_b}\|^2}. \tag{2.40}$$

The second phase of training involves using supervised learning, typically either standard BP or a variation of it, to adjust the output layer of weights; note that the hidden layer of weights are not adjusted.

2.6 Practicalities of Training Neural Networks

As indicated in the previous sections, there are many neural network parameters. Some are easily defined because they are dependent on external factors for a given application. For example, the number of inputs to the neural network directly relates to the nature of the input data. However, other parameters such as the learning rate and momentum values, or the number of hidden neurons, are more difficult to pre-define. Such parameters which have greater freedom in their chosen values may be used to optimise the neural network in terms of both its convergence speed, and its generalisation.

In the development of any neural network system, the optimisation of the convergence speed and generalisation is the main objective, and the associated parameter optimisation procedures are detailed in Section 2.7 and Section 2.8. One of the main practical issues which is dependent on the chosen neural network parameters is that of when to stop training.

2.6.1 When to Stop Training

In practice a neural network will be trained to an optimum state. Periodically throughout the training process, the weight values of the ANN will be saved together with testing output results and global error values. By having a series of 'snapshot' states of the ANN the task of pinpointing the optimum set of weights, determined by the optimum testing performance and generalisation ability of the neural network, is significantly easier.

A comparison of the global error values for both the training and testing data sets is a commonly used decision criterion. Figure 2.7 shows a typical plot of the training and testing global error values (SMSE is defined in Section 2.9.1) against epochs together with an indication of the region which gives optimum testing and generalisation performance.

2.6.2 Simulation Software

Several neural network simulation packages have been used throughout this work. Initially the standard BP neural network simulations were performed using PlaNet [75]

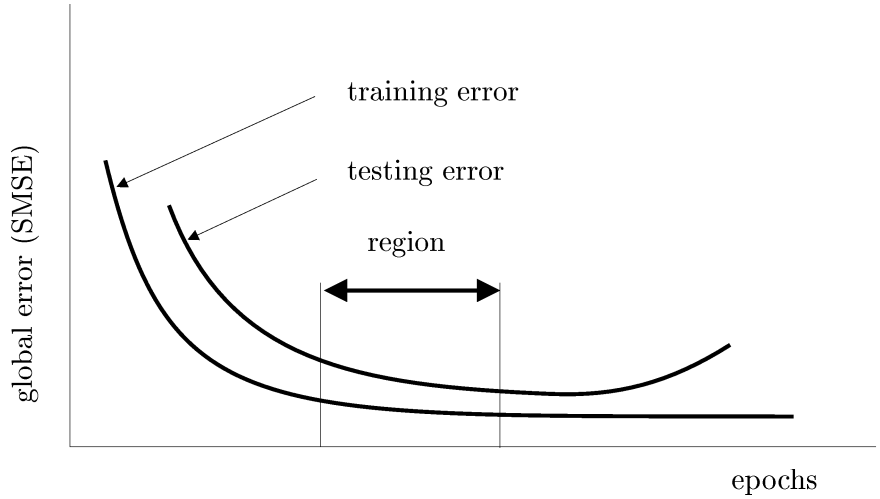


Figure 2.7: Training and testing error.

while the initial quickprop networks were performed with both Quickprop [76] and NevProp [77]. However, the NeuralWorks [78] simulator was chosen for the final simulations as it allowed simulation of many paradigms, including BP, QP, SOM and RBF.

2.7 Improving Neural Network Convergence Speed

The neural network parameters which directly affect the convergence speed are the weight initialisation, the learning rate and momentum values for BP (RBF and SOM are also affected by the number of patterns), the input and output data numerical ranges and the presentation order of the training data. These may also indirectly affect the generalisation.

2.7.1 Weight Initialisation

Before training can begin, the initial values of the weights need to be determined. For supervised algorithms random weights are used. Lari-Najaf *et al* [79] have shown that for maximum convergence speed, without reducing the generalisation performance, the initial weights should be bound in-between the range of -0.1 to 0.1. The SOM algorithm uses the input data to initialise the weights.

2.7.2 Training Parameters: Learning Rate and Momentum

The optimum learning rate and momentum values depend on the particular application, and although certain values have been suggested by researchers to be efficient, a heuristic search is the method favoured to find the parameter values which give optimum neural network convergence speed and generalisation performance. The momentum is the parameter which can lead to poor generalisation, since selection of a value which is too small may allow the global error to 'get stuck' in a local minimum on the error surface of weight-space.

2.7.3 Input and Output Data Ranges

In order to achieve maximum speed of convergence and to prevent the absolute values of the weights becoming too large, which affects the generalisation, producing a 'brittle' network, the numerical range of the input data should be normalised or rescaled to operate within the linear portion of the hidden layer activation function. For the sigmoid functions given in Figure 2.5, the typical ranges are 0 to 1 or -1 to 1.

For maximum performance the numerical range of the target classification data should be within the bounds of the output layer activation function being used. Hence, for the bipolar sigmoid of Figure 2.5(a), the range should not exceed the bounds of -1.0 to 1.0.

2.7.4 Presentation Order of the Training Data

For any given epoch, the presentation order of the training data is only effective in increasing the convergence speed if continuous updating of weights is used. Continuous updating is used in conjunction with a scheme for randomizing the order of pattern presentation from one epoch to the next, allowing the search in weight space to be stochastic over the learning cycles [80]. Typically periodic updating is used with small training data sets while continuous updating is most effective with large training data sets.

2.8 Optimising Neural Networks for Generalisation

The generalisation of a neural network refers to its ability to correctly classify the testing and validation data [81]. The parameters which affect the generalisation are the number of hidden neurons and training and testing data selection, assuming that the training solution has aquired the global minimum.

For good generalisation, Widrow [82] states that "the total number of (training) patterns should be several times larger than the neural network's capacity (defined as the number of weights in the network divided by the number of outputs)."[1] This is expressed as

$$N_p \gg \frac{N_w}{N_k} \tag{2.41}$$

where N_p is the number of training patterns, N_w is the number of weights and N_k is the number of outputs (output neurons). Assuming a fully connected architecture, then the number of weights may be defined in terms of the number of hidden neurons N_h, the number of input neurons N_i and the number of output neurons as follows

$$\begin{aligned} N_w &= (N_h \times N_i) + (N_h \times N_k) \\ &= N_h \left(N_i + N_k\right). \end{aligned} \tag{2.42}$$

Combining equations 2.41 and 2.42 gives

$$N_p \gg N_h \left(\frac{N_i}{N_k} + 1\right). \tag{2.43}$$

As the number of inputs and outputs are limited in their variation (details given in Section 2.8.1) and the total number of available training patterns may also be constrained (Section 2.8.2), the number of hidden neurons used must be carefully selected for good generalisation.

[1] Widrow [82] suggests this can be understood intuitively by noting that "if the number of degrees of freedom in a neural network (number of weights) is larger than the number of constraints (number of outputs multiplied by the number of training patterns), the training procedure will be unable to completely constrain the weights in the network."

2.8.1 Selecting a Neural Network Architecture

The neural network architecture refers to both the number of neurons in each layer and the number of layers, within the connectionist model. Typically, the number of neurons in the input layer will be determined by the dimensionality of the input data. The number of output neurons will vary on the required target classification (for supervised networks). For the majority of this work the target classification will be an output image, whose resolution will determine the required number of output neurons.

The number of hidden layers and the number of neurons in the hidden layers are less obvious to define. Work by several researchers has enabled the number of hidden layers required to map an arbitrary output function from a given input data set to be determined (Section 2.8.1.1). The specific number of neurons in the hidden layers will be dependent on the complexity of the input-output mapping and although many researchers employ an heuristic search method to determine the optimum number, work by Cover [83,84] and Baum [85] has given upper and lower bounds for the search (Section 2.8.1.2).

2.8.1.1 Number of Hidden Layers

Kolmogorov's theorem [86] which concerns the representation of arbitrary continuous functions from the n-dimensional cube to the real numbers in terms of one dimensional functions, forms the basis upon which improvements, discovered by Lorentz [87] and Sprecher [88], lead to the reinterpretation by Hecht-Nielsen to give Kolmogorov's Mapping Neural Network Existence Theorem which states [89]:

> "Given any continuous function $\phi : I^n \longrightarrow R^m$, $\phi(x) = y$, where I is the closed unit interval [0,1] (and therefore I^n is the n-dimensional unit cube), ϕ can be implemented *exactly* by a three-layer (one hidden layer) neural network having n processing elements in the input layer, $(2n + 1)$ processing elements in the hidden layer and m processing elements in the output layer"

Thus, theoretically a neural network with one hidden layer, containing $(2n+1)$ neurons, can map any n-dimensional input data (x) onto any output (y). It has been concluded by other researchers, including Villiers and Barnard [90], that one hidden layer is sufficient for all except the most esoteric applications.[2] Villiers and Barnard also show that there is no performance benefit gained by using two hidden layers in preference to one.

2.8.1.2 Number of Hidden Neurons

Although Kolmogorov's theorem as stated by Hecht-Nielsen [89] specifies the maximum number of neurons in the hidden layer to map any input and output data together, this theorem assumes that the neural network maps the n-dimensional input data by a series of superpositions of one-dimensional functions [93].[3]

Studies by Cover [83,84] on the capacity of a MLP network using a signum activation function determined the lower bound on the minimum number of weights needed to realize any Boolean function. Baum [85] extended this work for multi-output networks and also finds a corresponding upper bound (for one hidden layer networks):

$$\frac{N_k N_p}{1+\log_2(N_p)} \leq N_w < N_k\left(\frac{N_p}{N_i}+1\right)(N_i+N_k+1)+N_k \tag{2.44}$$

where N_p is the number of training patterns, N_i refers to the number of input neurons, N_k is the number of output neurons and N_w is the number of weights in the neural network (excluding the bias weights).

Again assuming a fully connected architecture, substituting equation 2.42 into expression 2.44 gives

$$\frac{N_k N_p}{(1+\log_2(N_p))(N_i+N_k)} \leq N_h < \frac{N_k\left(\frac{N_p}{N_i}+1\right)(N_i+N_k+1)+N_k}{(N_i+N_k)}. \tag{2.45}$$

In practice, having too many hidden neurons causes the neural network to learn a one-

[2] Some applications may require additional filtering or mapping to be performed by a second hidden layer [91,92]

[3] However, this is not the case, and so the neural networks' maximum performance may be achieved with far fewer hidden neurons than required by Kolmogorov's theorem.

to-one mapping of the training data, acting as a look-up table, and reducing its ability to generalise for unseen data. Having too few hidden neurons will prevent the neural network from converging properly. The range defined by equation 2.45 is thus used as a guide for setting the bounds of the heuristic search to determine the optimum number of hidden neurons.

2.8.2 Training and Testing Data Selection

In order to achieve good generalisation from a neural network, the selection of training data samples which fully represent the input-output mapping space (problem domain) is essential. Practical constraints on the data collection procedure typically limit the total number of available data samples. Given a database of all the available data samples, selection of appropriate samples for training and testing data should be performed. In doing so, it is imperative that the data samples selected for the testing data are not used in the training data. This limitation may prevent the formation of a representative training data set, generating unequally represented target classes.

It has been shown by DeRouin and Brown [94] and Anand *et al* [95], amongst others, that a neural network will become biased towards a target class which has more exemplars in the training data (assuming that the target classes are unequally represented in the training data). This suggests that a neural network utilizes the probability distribution of the target classes within the training data and concentrates its learning on the class with greatest probability. However the validation data is unknown during the training process and so the *a priori* probability distributions of the validation data target classes are also unknown and may not be assumed to be the same as for the training data. At best, the application of the principle of indifference implies that the *a priori* probability of each target class in the validation data is the same. This must then be reflected in the training data; several methods exist to reduce the problems of unequally represented target classes:

- Experimental design of the data collection procedure may be possible to ensure that all target classes are equally represented in the training and testing data.

- Duplication of the under represented target classes to give the same number of exemplars of each class.
- DeRouin and Brown [94] suggest a method of modifying the learning rate in order to compensate for unequally represented classes.

2.9 Measuring Neural Network Performance

The neural network's performance needs to be measured for two reasons, the first is to determine when the neural network has "finished" training and the second is to allow comparisons between several different neural networks. Several performance parameters have been used, Section 2.9.1 details the parameter used to determined when to stop training a neural network, while Section 2.9.2 to Section 2.9.5 give the various parameters used to compare neural network performance.

2.9.1 Sum of Mean Squared Error (SMSE)

Here the SMSE is the value of the global error per output neuron per pattern for a given epoch, and as demonstrated in Section 2.6.1 (Figure 2.7) is used to determine when to stop training a neural network.

$$\text{SMSE} = \frac{1}{2KP} \sum_{p=1}^{P} \sum_{k=1}^{K} (t_{pk} - o_{pk})^2 \tag{2.46}$$

2.9.2 Performance Measures

In the simplest case, assume that each output neuron is used to classify between two classes, then for a given neural network decision, indicated by the output state of a given output neuron after appropriate input data are presented to the network, four possible alternatives exist [96], as given in Table 2.1.

The first alternative is a true positive (TP), in which the positive diagnosis of the output neuron coincides with a positive result according to target standard. For example, the neural network identifies the presence of a defect (positive diagnosis) at the location represented by the responding output neuron, in agreement with the user

Table 2.1: Two-class classification categories.

Output Neuron	Target Standard	
	positive	negative
positive	true positive TP	false positive FP
negative	false negative FN	true negative TN

defined reality (target standard). The second is a false positive (FP), where the output neuron made a positive diagnosis which is incorrect according to the target standard. In the third alternative, a false negative (FN), the output neuron indicated the absence of a positive diagnosis which was incorrect according to the target standard. The last definition is a true negative (TN), in which the negative diagnosis of the output neuron agrees with the target standard.

For any given trained neural network, the testing procedure involves presenting the neural network with testing input data and collecting the resulting outputs from the output neurons of the neural network. The collected outputs are then compared with the corresponding target values, giving for each output and target a categorisation as defined in Table 2.1.

The comparison requires a threshold value which specifies the acceptable range of values from each output neuron associated with a positive and negative diagnosis. Typically the threshold would reflect the midpoint between the two target standards and is referred to as the middle threshold. For example, if a positive diagnosis is to be represented by an output value of 1.0 and the negative diagnosis by an output value of 0.0 then the middle threshold would be 0.5. Similarly if a positive diagnosis is represented by a value of 1.0 and a negative diagnosis by a value of -1.0 then the threshold would be set at a value of 0.0.

Calculating the occurrence of each of the four output/target categories (TP, FP, FN, and TN) over the complete testing data allows the numerical values of the five performance metrics to be determined using the definitions, taken from Eberhart and Dobbins [97], given below. The ranges and ideal values for the five performance metrics are shown in Table 2.2.

Table 2.2: Values for the performance metrics.

Performance Metric	Value Range	Ideal Value
Sensitivity	0.0 to 1.0	1.0
Specificity	0.0 to 1.0	1.0
Positive Predictive Value	0.0 to 1.0	1.0
False Alarm Rate	0.0 to 1.0	0.0
Percentage Correct	0 to 100	100

2.9.2.1 Sensitivity

The Sensitivity (Sens) of a neural network indicates the likelihood of an event, for example the presence of a defect, being detected given that it is present.

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.47}$$

2.9.2.2 Specificity

The Specificity (Spec) of a network is the likelihood that the absence of an event will be detected given that it is absent, and is given by

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}. \tag{2.48}$$

2.9.2.3 Positive Predictive Value

The Positive Predictive Value (PosPV) is the likelihood that a signal of an event is associated with the event, given that the signal occurred, and may be defined as

$$\text{Positive Predictive Value} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.49}$$

2.9.2.4 False Alarm Rate

The False Alarm Rate (FalAR) is the likelihood that a signal is detected falsely given that a non target event occured, where

$$\text{False Alarm Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{2.50}$$

2.9.2.5 Percentage Correct

The Percentage Correct (% Correct) metric evaluates the percentage of correctly assessed diagnoses by a neural network, and is written as

$$\text{Percentage Correct} = 100 * \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{2.51}$$

2.9.3 Threshold Bounds defining Uncertainty

For the previous performance measures, a single threshold upon the activation of a given output neuron is used as the boundary between a positive and negative classification. This is sufficient for testing but for validation data an additional 'uncertain' classification is desired and is achieved by having two threshold boundaries to give three classification regions, negative, uncertain and positive, as shown below.

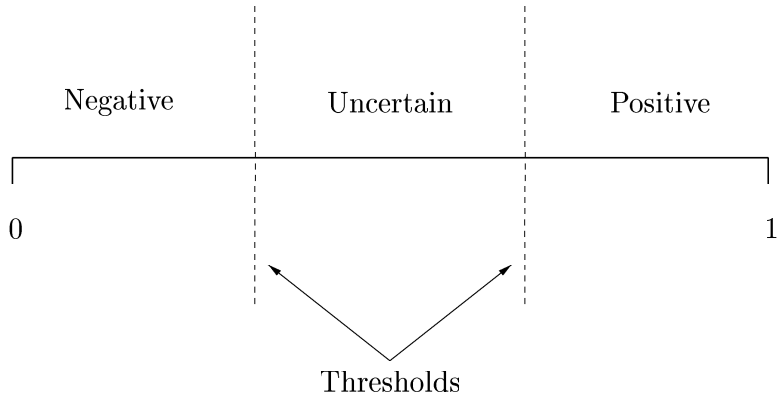


Figure 2.8: Classification regions and threshold boundaries for an output neuron.

Selection of the exact position of the two thresholds along the numerical range of the output neuron is determined by re-calculating the TP, TN, FP and FN values for various single threshold values using the testing data. Identification of the upper and lower threshold values which give reduced testing performance, from the maximum, will specify the two thresholds for the uncertain classification to be used with the validation data.

It is the author's view that the larger the region of uncertainty, defined by the threshold bounds, the greater the generalization performance for validation data.

2.9.4 Confusion Matrix

A confusion matrix allows examination of the neural networks ability to classify the testing or validation data. In particular it illustrates all of the positive (defect) classifications made by the neural network for a given data set. Consider the matrix shown in Figure 2.9 where the number of rows and columns of the matrix are always equal and represent the different classifications possible for a given data set, typically defect location. The columns indicate the target classification and the rows give the actual neural network classification.

The main diagonal relates to examples which have been correctly classified by the neural network as a defect, a true positive diagnosis. Any other elements within the matrix represent false positive classifications (the position of the element within the matrix details both the target defect location of the example and the actual location classification given by the neural network).

For all confusion matrices, the middle threshold value, typically 0.5, is used to determine if the actual classification is positive or negative.

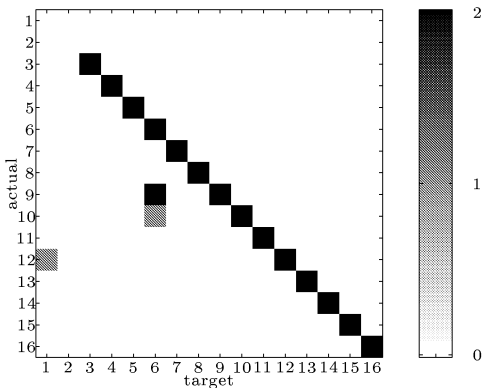


Figure 2.9: A confusion matrix.

0.00	1.00	0.00	1.00
0.00	1.00	0.50	1.00
1.00	1.00	1.00	1.00
1.00	1.00	0.50	1.00

Figure 2.10: A confidence map.

2.9.5 Confidence Map

The confidence metric uses the confusion matrix, generated from testing data, to indicate the degree of confidence of correct defect detection at each possible location for the validation data.

The confidence metric for each classification c, is calculated as follows

$$\text{Confidence}_c = \frac{\text{Confusion}_{cc}}{N_{e_c}} - \frac{\sum_{d \neq c} \text{Confusion}_{cd}}{N_{mc}} \tag{2.52}$$

where N_{e_c} is the total number of examples of target classification c in the testing data set and N_{mc} is the maximum number of mis-classifications for any given target class. This will give a value in the range of between -1 and 1 where a value of 1 indicates that the given output correctly identified its own class examples and was never stimulated by examples of other classes while a value of -1 warns that both the output's own class examples were not identified and the maximum number of examples of other classes incorrectly stimulated the output.

Because most of the neural network training within this thesis relates to defect location within a scanning area, the confidence metric can be formed into a confidence map, illustrating portions of the scanning area the neural network is most likely to correctly identify defects. An example is shown in Figure 2.10.

2.10 Summary

In this chapter, the neural network learning algorithms used throughout the present work have been developed. Most of the present work uses the BP algorithm or a variation of it, selection of which centres on the apparent popularity of BP for research applications. In addition to the NN theory, practical issues concerning the development of a trained neural network system have been described. These mainly involve the convergence speed and generalisation ability of the neural network and as shown, can be optimized with correct selection of various parameters.

Chapter 3

Instrumentation for Tomographic Ultrasonic Data Collection

3.1 Introduction

Instrumentation has been developed for the transmission and reception of ultrasonic waves, to be used with the basic geometry outlined in Chapter 1, illustrated in Figure 1.3 and referred to as the sensor array throughout this thesis. The implementation of the instrumentation used for the collection of tomographic data from the sensor array depends on the actual number of sensors acting as transmitters and receivers of ultrasound within the sensor array. Hence, as will be described, two different implementations have been used. The first is used in conjunction with a sensor array consisting of 16 sensors, specifically 12 transmitters and 4 receivers, while the second uses an array of 40 sensors, 32 acting as transmitters and 8 as receivers.

Although for practical purposes, selection of the composite material used for the samples for data collection will ultimately depend on the type of composite material to be used for a given application, the criteria for the selection of the composite materials used within this thesis were arbitrary and relied more on the availability. Thus, the composite material used for the majority of the experimental samples was a glass fibre reinforced composite supplied by Fibreforce Composites Ltd., Runcorn, England. Details of this material plus others used are given below. Determining the ultrasonic wave modes present within the various composite materials was empirical; due to the highly

anisotropic nature of the composite materials, no single wave mode may be assumed and it was anticipated that several modes would be present.

The experimental apparatus used throughout this work consists of two main components: the sensor array, which houses the ultrasonic sensors, and the instrumentation which collects the tomographic data from the sensor array. Section 3.2 describes the sensor array while Section 3.3 details the two implementations of the instrumentation. This is followed by a description of the dedicated instrument control software developed for the second implementation of the instrumentation. Then a description of the various composite materials used is given together with analysis of their ultrasonic behaviour. This is followed by details of the implementation of the final system including the integration of the developed neural network system.

3.2 The Sensor Array

The sensor array consists of a square frame Perspex holder, containing a number of ultrasonic transducers which are positioned on all four sides of the square frame, as shown by Figure 3.1. The Perspex holder allows the transducers to be held in a vertical position directly above the material sample. The physical size of the sensor array is such as to give a scanning area of 80mm by 80mm.

The specific type of ultrasonic transducers used are called pinducers, detailed in Section 3.2.1 and can act as either a transmitter or receiver of ultrasound. A given sensor array contains a specific number of pinducers. As will be seen, it is easier to have a fixed number of pinducers within the sensor array solely acting as receivers and to multiplex the source position using the remaining pinducers. Hence the transducer configuration of a given sensor array refers to the exact number of pinducers within the sensor array acting as a receiver or a transmitter of ultrasound. The receiver arrangement specifies the exact location of the receivers within the square geometry of the sensor array.

Although Figure 3.1 shows the sensor array with 16 transducers, four on each side, the square frame holder is able to contain up to 40 transducers, ten on each side. The specific transducer configurations and receiver arrangements are detailed in Section 3.2.3.

3.2.1 The Ultrasonic Pinducers

The miniature ultrasonic transducers, known as pinducers and manufactured by Valpey-Fisher[1], have an active area of 1mm diameter and operate at a centre frequency of 1.8MHz. The pinducers are piezoelectric devices, and are constructed by taking a stainless steel casing and inserting a brass rod which is insulated by an epoxy filler (Figure 3.2 shows a cross-section). The exposed end of the rod makes electrical contact, via a pinducer socket, to a pulser. The other end is in direct contact with a disc of lead zirconate titanate (PZT) which forms the active end of the pinducer, the bottom surface of which is coated with an earthed thin layer of gold, deposited via evaporation.

3.2.2 Pinducer Coupling

A practical problem with contact transducers is the difficulty of ensuring constant ultrasonic coupling between the transducer and the inspection sample throughout an experiment. Both the physical boundary between the surface of the transducer and the sample and the amount of pressure forcing the transducer to make contact with the sample are of consideration.

Initial experiments using the 16 transducer sensor array used a high viscosity liquid couplant and relied on the weight of the square frame to supply the downward force on the pinducers. However, both the volume of liquid couplant used for each pinducer and the friction between the square frame and the pinducers cannot be assumed to be constant throughout an experiment.

Two improvements to the pinducer coupling have been used with the 40 transducer sensor array. The first employs a spring and collar mechanism. Illustrated in Figure 3.3, a collar is attached to each pinducer just above the active end with a spring placed between the collar and the lower surface of the square frame. With additional weight used to force the frame down towards the sample, the pressure upon each pinducer can be assumed to be constant throughout an experiment.

The second improvement uses a silicone rubber sheath placed on the active end of each pinducer which allows a more uniform coupling between the pinducer and the

[1]Hopkinton, MA, USA.

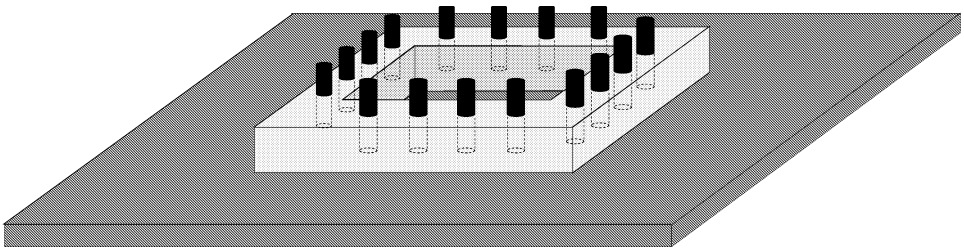


Figure 3.1: 3D view of the basic geometry.

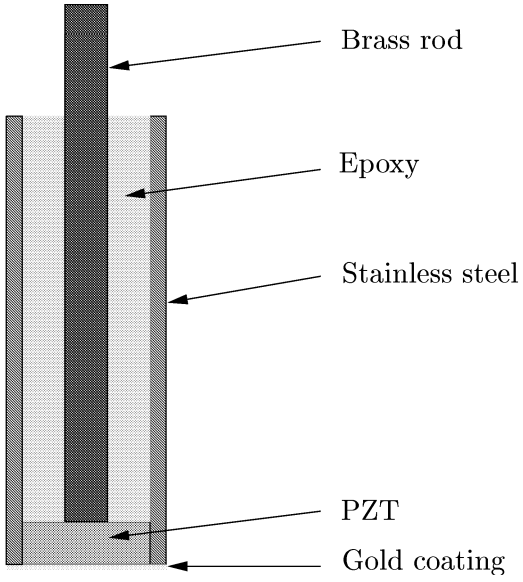


Figure 3.2: Pinducer construction.

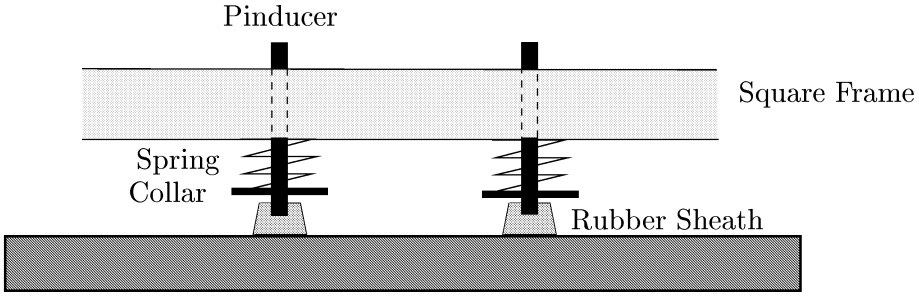


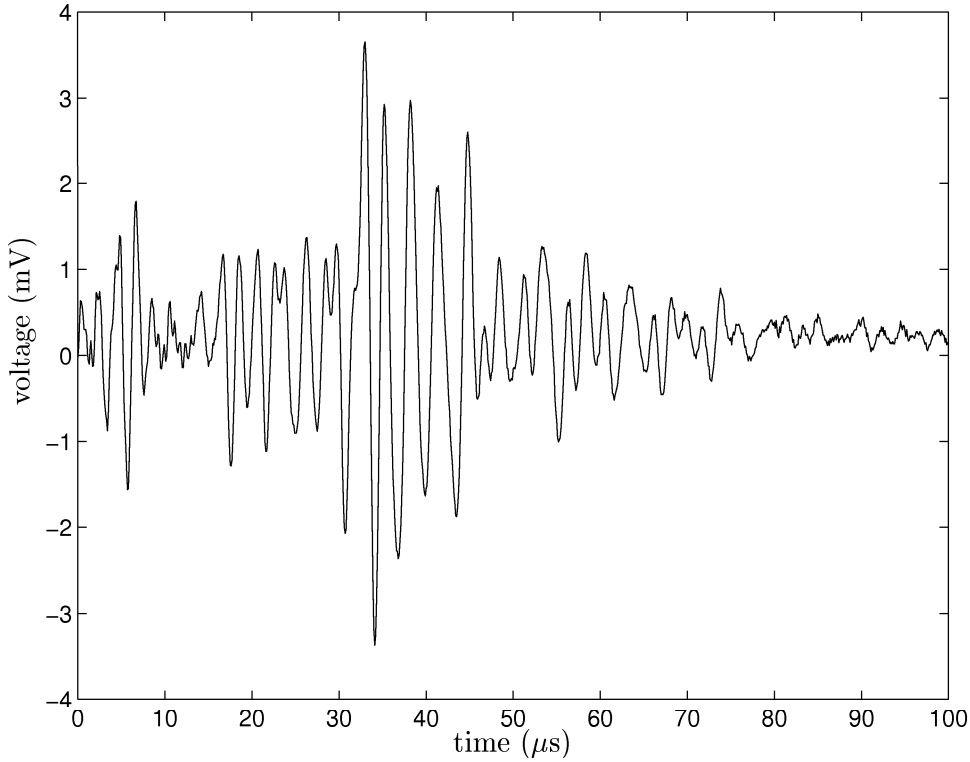
Figure 3.3: Side view of two pinducers with coupling improvements.

sample throughout an experiment. Comparative ultrasonic waveforms, both with and without the rubber coupling are given in Figure 3.4, for a pair of pinducers spaced 50mm apart perpendicular to the unidirectional fibre bundles of the glass fibre-reinforced composite detailed in Section 3.5. Figure 3.4 illustrates that there are significant differences between the two time domain waveforms when using a different transducer couplant. However, Figure 3.5 shows that the frequency components within the two time domain waveforms have similar envelopes. In fact over 85% of the energy within the time domain waveforms are contained within the same frequency range (0.1MHz to 0.7MHz). Because neural networks are to be used any minor differences between the two waveforms would be learnt.

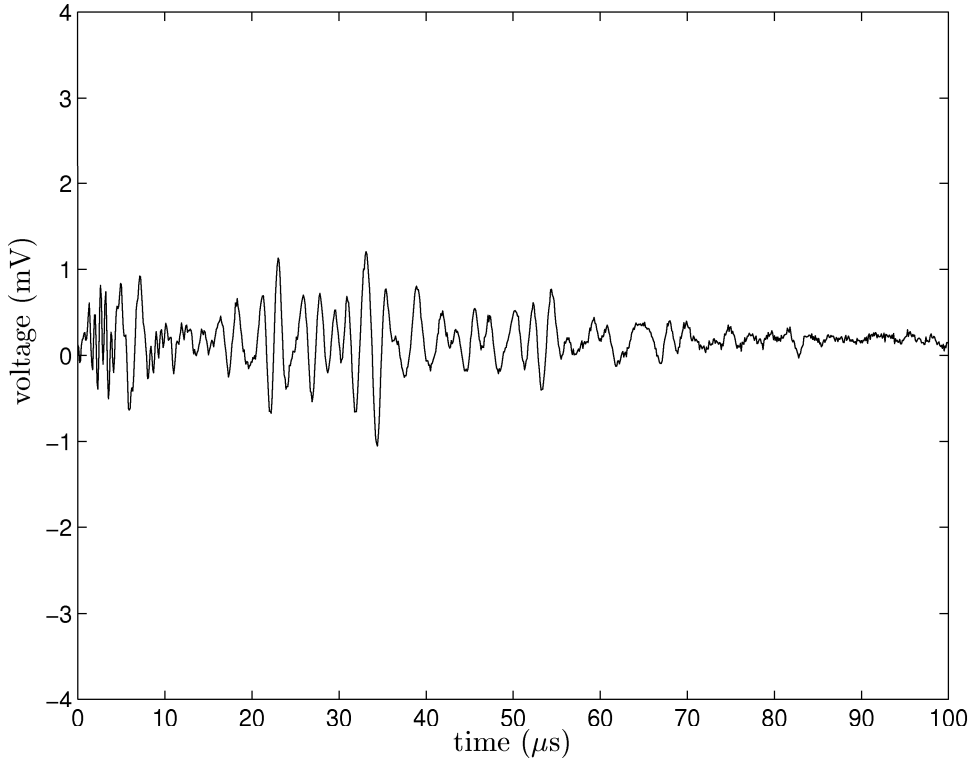
3.2.3 The Two Transducer Configurations

Two different transducer configurations have been used throughout this work and are shown in Figure 3.6. The number of transducers used for the two configurations was constrained by the limitations of the experimental equipment available. However, the effectiveness of using 16 and 40 transducers with a scanning area of 80mm by 80mm was initially investigated by simulation (see Section 4.3) to determine the minimum defect size detectable.

The 16 transducer sensor array, Figure 3.6(a), has been configured to have 4 receivers and 12 transmitters, while the 40 transducer sensor array, Figure 3.6(b), has been configured to have 8 receivers and 32 transmitters. For the 16 transducer configuration two receiver arrangements have been used and they are shown in Figure 3.7 (the receivers are denoted by white circles). The first arrangement, Figure 3.7(a), was arbitrarily chosen. However, the second, Figure 3.7(b), was selected to improve the raypath spatial density within the sensor array. As shown in Figure 3.8, arrangement B has a more even raypath coverage or density than arrangement A. The receiver arrangement for the 40 transducer configuration is determined by computer simulations due to the large number of possible arrangements of the eight receivers and details are given in chapter 4.

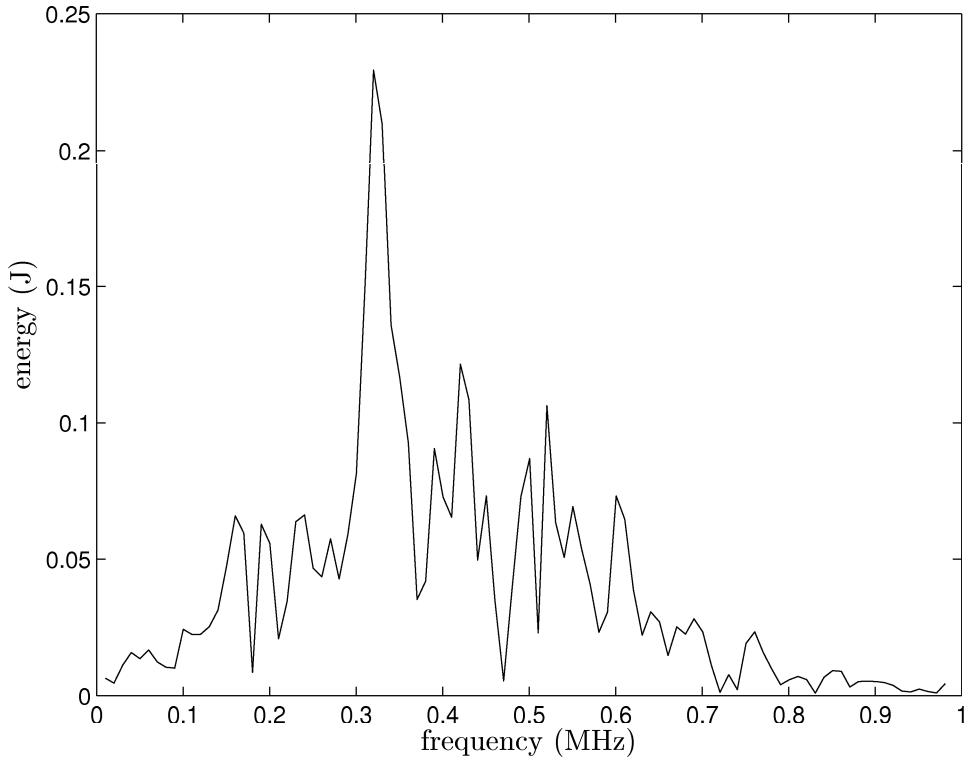


(a) with liquid coupling.

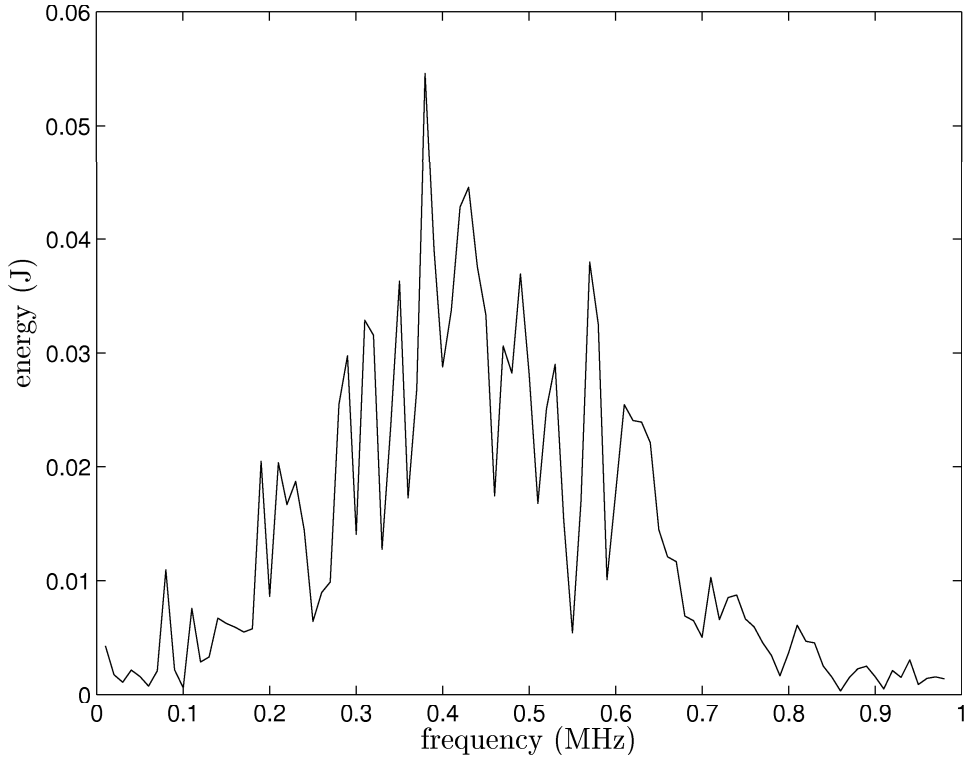


(b) with rubber coupling.

Figure 3.4: Comparison of the pinducer coupling methods (time domain).

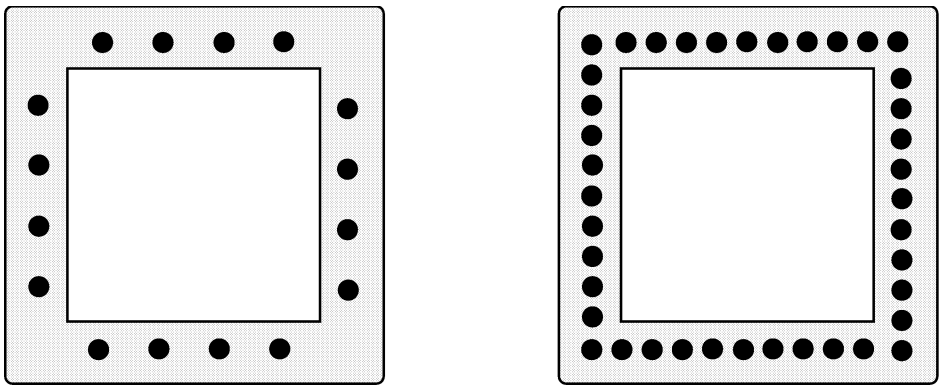


(a) with liquid coupling.



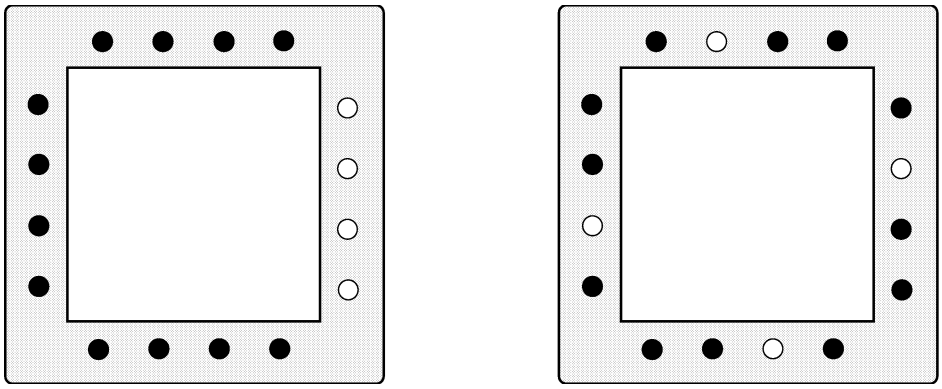
(b) with rubber coupling.

Figure 3.5: Comparison of the pinducer coupling methods (frequency domain).



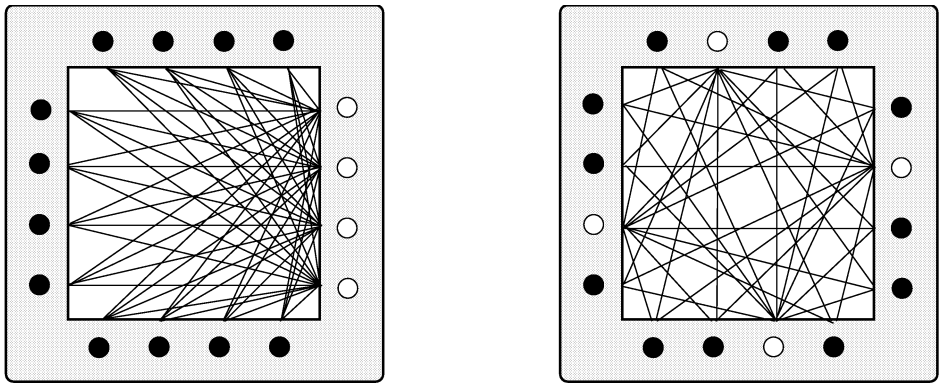
(a) 16 transducers. (b) 40 transducers.

Figure 3.6: Top view of the square frame showing the two sensor arrays.



(a) Arrangement A. (b) Arrangement B.

Figure 3.7: The two arrangements for the 16 transducer configuration.



(a) Arrangement A. (b) Arrangement B.

Figure 3.8: Raypaths for the two arrangements of the 16 transducer configuration.

3.3 Instrumentation for Multiplexing

The instrumentation required ultimately depends on the number of transducers within the sensor array, specifically on the exact number of receivers and transmitters within the sensor array. As indicated by Section 3.2.3 there are two transducer configurations, and thus two different implementations of the instrumentation. Although the ultrasonic pinducers may act as both a receiver and transmitter of ultrasound, the sensor arrays are defined with specific pinducers permanently acting as either a receiver or transmitter, thus reducing the complexity of the instrumentation.

3.3.1 Instrumentation for the 16 Transducer Configuration

The instrumentation used with the 16 transducer configuration is shown in Figure 3.9. The pinducers acting as transmitters are switched by a Keithley 705 multiplexer allowing each pinducer in turn to be excited by a Panametrics 5055PR signal pulser[2]. Four pinducers are configured to act as receivers, the signals being captured by a 4-channel Nicolet 460 digital oscilloscope after amplification with a Cooknell CA6 charge amplifier.

Manual operation of the Keithley multiplexer and the Nicolet oscilloscope allow the collection of tomographic data. The oscilloscope was configured to perform signal averaging to reduce the effects of noise, with each received signal being averaged 128 times before the resulting waveform is saved. Due to the manual operation of the equipment several seconds elapsed between the collection of each waveform, allowing the material sufficient time to become ultrasonically silent. The pulse repetition frequency of the Panametrics signal pulser was set at 50Hz. The waveforms are stored as a series of 2,000 single precision floating point numbers. Note that arrangement A (Figure 3.7(a)) gives 48 waveforms per tomographic scan while arrangement B (Figure 3.7(b)) gives 36 waveforms.

[2]The Panametrics generates a pulse of amplitude -250 Volts, a rise time of 10ns and a pulse repetition frequency of 50Hz.

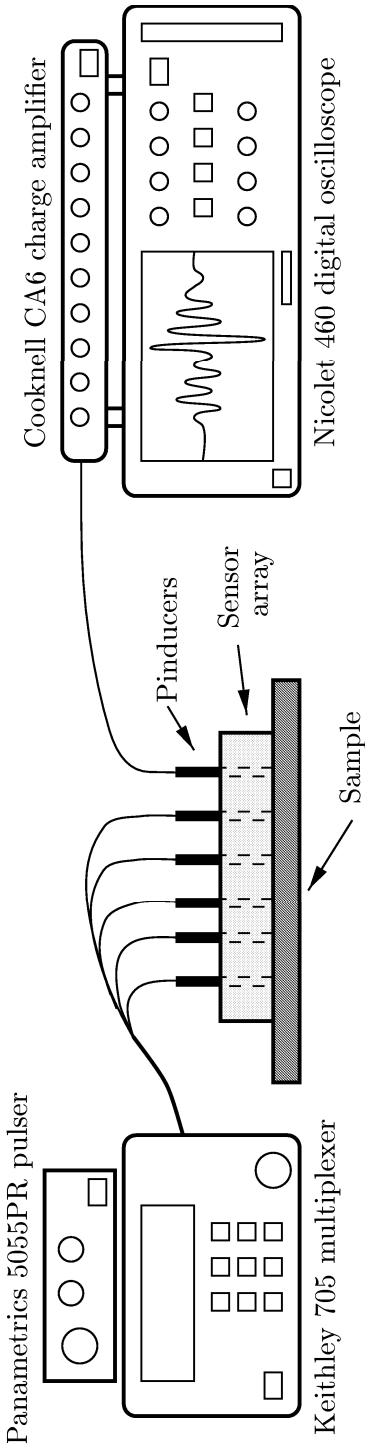


Figure 3.9: Experimental apparatus for the 16 transducer configuration.

3.3.2 Instrumentation for the 40 Transducer Configuration

The design strategy for the 40 transducer configuration was to develop an instrumentation system that was completely self-contained within a personal computer. This led to the system described in Section 3.3.2.1 being developed. However initial evaluation concluded that while the system would work for metals, the inspection of composite materials was unachievable due to lack of output pulse amplitude from the pulser. Thus, another instrumentation system was developed which improved system performance by using an external pulser and oscilloscope, allowing the inspection of composite materials. A description of the modified instrumentation is given in Section 3.3.2.2.

3.3.2.1 Self-contained Instrumentation

An IBM computer was installed with a SMIS[3] S1200 ultrasonic pulser card (output amplitude of -100V and a rise time of 12ns), a SMIS S1140 two channel ultrasonic receiver and digitiser card and a Keithley PIO-32 series 32-channel multiplexer card. Given these three computer cards, control software, detailed in Section 3.4, was written to allow manipulation of their individual functions. This allowed extensive investigation of the capabilities of both the individual cards and an evaluation of the performance of combining the operation of several cards. This study demonstrated that (a) due to the power limitations of the pulser's output and (b) the additional noise experienced from within the computer environment, affecting the multiplexer attached to the receiver card, the inspection of composite materials could not be achieved with this completely self-contained system. However, it produced good results in metallic plate samples, and was considered a success in this regard.

3.3.2.2 Modified Instrumentation for Composite Materials

The instrumentation used with the 40 transducer configuration is shown in Figure 3.10. The transducers acting as transmitters are connected to the Panametrics 5055PR signal pulser via a Keithley PIO-32 series multiplexer card controlled by the IBM computer allowing each transmitter in turn to be excited. The IBM computer also had a GPIB

[3]Surrey Materials Inspection Systems Ltd.

IEEE 488 interface bus which controls and transfers data from the 4-channel Nicolet 460 digital oscilloscope which is connected to the transducers acting as receivers via the Cooknell CA6 charge amplifier. The diagram also shows the sensor array with the coupling improvements previously described.

Instrument control software, detailed in Section 3.4, automates the operation of the Keithley 32-channel multiplexer card together with the Nicolet oscilloscope via the GPIB IEEE 488 interface card and transfers the received waveforms from the oscilloscope to the computer's hard disk. Again the signals were averaged 128 times before being saved and the waveforms were stored as a series of 2,000 single precision floating point numbers. The receiver arrangement detailed in chapter 4 generates 184 waveforms per tomographic scan.

Note that there is a difference between the number of waveforms generated with the experimental arrangement using 40 transducers compared with the simulated arrangements given in Section 4.3. This is due to the experimental arrangements having tranducers on the corners, which are effectively on two sides of the square array, and so reduce the total number of raypaths which travel through the scanning area.

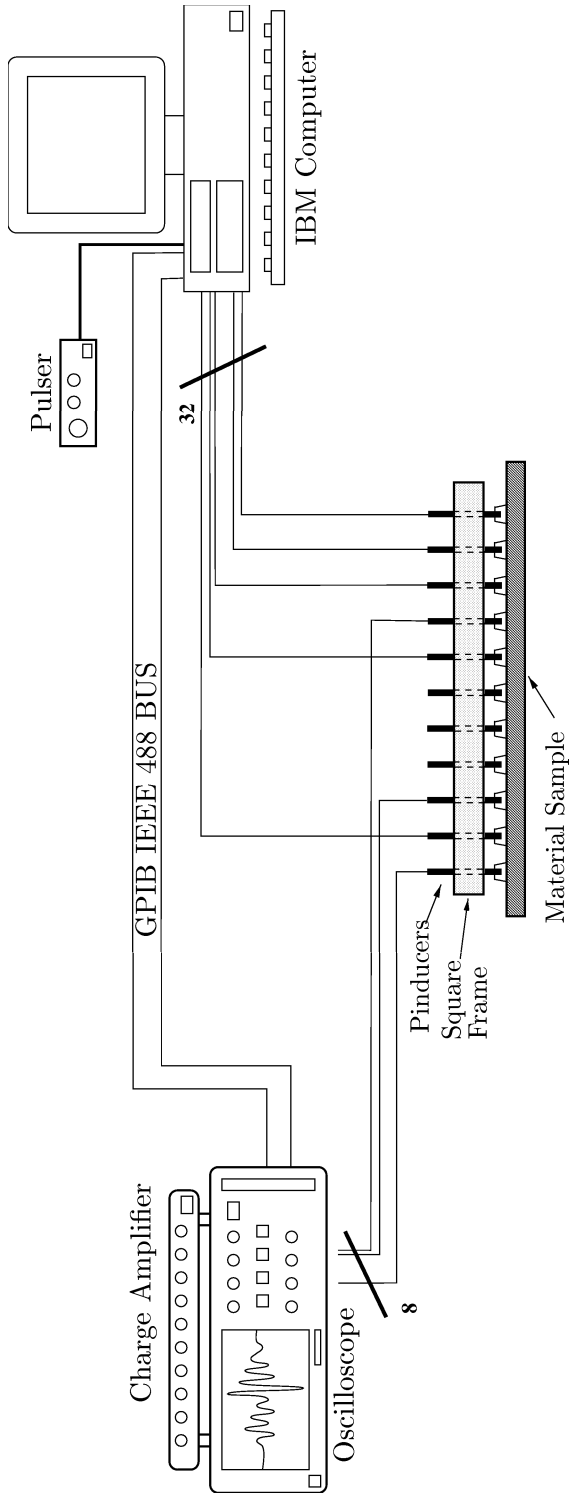


Figure 3.10: Experimental apparatus for the 40 transducer configuration.

3.4 Instrument Control Software

Dedicated instrument control software has been written using the C++ programming language for the Microsoft Windows operating system. Two versions of the software have been developed, both for the instrumentation using the 40 transducer sensor array. The first version controls the self-contained instrumentation, given in Section 3.3.2.1, which includes a pulser, receiver and digitiser and multiplexer computer cards. The control software code is given in appendix B. It gives independent functional control of all three PC cards, serving as an interface between the user and the specific operations of each card. Such flexibility in control allowed the evaluation of the system for the inspection of various materials.

As previously described, the self-contained instrumentation was ineffective for the inspection of composite materials, and this lead to the development of the modified instrumentation and to the second version of the control software. The second version of the software controls the multiplexer card and the 4 channel digital oscilloscope via the IEEE GPIB interface card. The algorithmic operation of this version of the software is given in Table 3.1 as pseudo code, while the actual program code is listed in appendix B. Here the software automates most of the data collection process, including switching of the pulser's signal to the transmitting pinducers via the multiplexer card, the capture of the received waveforms using the oscilloscope and the transfer of the waveforms to the computer's hard disk. The software initially requires the user to specify the relative positioning of the 8 receivers within the sensor array, allowing waveforms, generated by transmitter and receiver pairs which are both on the same side of the sensor array, to be ignored as they have not travelled through the scanning area. Note that because the oscilloscope has only 4 channels and the transducer configuration specifies 8 receivers, the software requires the user to change the 4 receivers being used to collect waveforms half way through each tomographic scan.

Table 3.1: Control of the modified instrumentation.

```
Require: positions of the eight receivers within the sensor array are known
Ensure: the eight receivers are grouped into two sets of four
  initialise IEEE GPIB interface
  initialise multiplexer and oscilloscope
  for both sets of receivers do
    for all of the transmitters do
      switch old active transmitter off {via multiplexer}
      settle delay
      switch new active transmitter on
      operate oscilloscope to collect all four waveforms
        for each of the four receivers do
          if receiver is not on the same side of sensor array as the transmitter then
            transfer that receiver's waveform to computer hard disk
          else
            ignore receiver's waveform
          end if
        end for
      end for
    prompt user to connect the next set of receivers
  end for
```

3.5 The Composite Samples

Two different composite materials are used throughout this work, the first is a glass fibre reinforced composite while the second is a carbon fibre reinforced composite. Following a description of manufacture of each of the composite materials, details of the ultrasonic response are given with varying propagation direction.

3.5.1 The Glass Fibre Reinforced Composite Material

Two slightly different glass fibre reinforced composite (GFRC) materials were used. The first, referred to as GFRC(a), was produced from a material called EXTREN[4] 525, and made by the pultrusion process [14]. The reinforcement used was E-glass fibre continuous filament mat (CFM) to impart some transverse strength and stiffness and E-glass unidirectional roving bundles to provide the essential longitudinal properties (the amount of each reinforcement type is unknown as the manufacturer considers such information proprietary). The notionally uniform distribution of longitudinal roving

[4]EXTREN is a registered trademark of Morrison Molded Fibre Glass (MMFG) Co., Bristol, Va.

bundles were sandwiched between outer and inner layers of CFM. The unidirectional fibre-reinforcing bundles do not form a continuous layer as the pultrusion process forces the bundles to become elongated in the transverse direction, hence the spacing between bundles was not constant and varied between 1 and 2 cm. The two fibre reinforcements were supported in a matrix consisting of an isophthalic polyester resin mixed with 10-15% by weight of a calcium carbonate filler.

The second glass fibre material, GFRC(b), used EXTREN 500 which has the same layup (reinforcement composition) as the 525 series, and again uses an isophthalic polyester resin. However, it has slightly different physical properties, such as non-flame retardance, but is assumed not to affect the ultrasonic behaviour compared with GFRC(a).

3.5.2 The Carbon Fibre Reinforced Composite Material

The carbon fibre reinforced composite (CFRC) material used was 16 ply unidirectional laminate of 2mm thickness. This was laid up from sheets of carbon fibre and epoxy pre-preg. All sixteen fibre layers were aligned so that the fibre orientation was the same for all layers. This was then cured at 120°C for two hours in a headed compression mould. The nominal fibre content was 60% by volume [10].

Simulated defects were introduced during the manufacture process by the inclusion of either teflon or chopped fibre square patches buried beneath the surface. Square patches of varying size (50, 25, 12 and 6mm) were cut out of the middle eight plies of laminate before they were laid up. After stacking the first twelve plies with the top eight having square holes, eight squares of 125μm thick Teflon film of the appropriate area were inserted and finally the remaining four plies stacked on top covering the filled holes (pits). For the chopped fibre defects, square patches of size 25mm but with varying thickness, including 2, 6, 8 and 12 plies thick, were cut out of the middle plies of the laminate during manufacture. Layers of randomly chopped fibres were instead inserted and the remaining complete plies stacked on the top covering the filled patches.

3.5.3 Ultrasonic Behaviour

The variation in ultrasonic response with changing propagation direction is illustrated for the two main composite types. Two pinducers, one acting as a receiver the other as a transmitter were spaced 50mm apart and rotated by the midpoint of separation through 90° at intervals of 10°. Both the glass fibre, GFRC(a), and the carbon fibre materials were used. For both the 0° direction refers to the perpendicular to the main fibre direction and the 90° direction is parallel to the main fibre direction.

Figure 3.11 illustrates the ultrasonic waveforms collected from the carbon material. As expected the ultrasonic signal has maximum amplitude and fastest arrival time (time of flight) when travelling parallel to the fibre direction. Because the fibres are only aligned in the one direction both the travel time appears to increase and the amplitude decrease when the propagation direction tends away from the fibre direction. The waveform collected at 90° shows a prominent response just after triggering but before the main peak to peak transient. This is assumed to be due to the presence of continuous carbon fibres between the two transducers allowing different wave modes to be detected.

Figure 3.12 shows the ultrasonic waveforms collected from the glass composite. The largest amplitudes are achieved with travel directions both parallel and perpendicular to the fibre direction. However, unlike the carbon material, both the amplitude and arrival times don't appear to conform to a simple relationship with travel direction.

The glass fibre composite material has a very complex internal structure which significantly complicates the intrepretation of these waveforms. The unidirectional glass fibres are arranged into a number of bundles of fibres. Each bundle is seperated from the others by the calcium carbonate filler. Hence, the waveform collected at 90°, shown in Figure 3.12, might have travelled along either a bundle of glass fibres or the composite filler. This might account for the absence of a transient just after triggering (which is evident for all other direction angles).

For the other propagation directions shown in Figure 3.12 another aspect of the material's internal structure is thought to be influential. The continuous filament mats, which sandwich the unidirectional bundles, will aid the propagation of ultra-

sound through the material along non parallel directions. However, due to the random nature of these glass fibre mats, it is anticipated that the effects of ultrasonic reflection, mode converion and scattering will be evident and account for the reduced clarity of these waveforms.

Figure 3.11 and Figure 3.12 have illustrated the anisotropic nature of both the carbon and glass fibre material. Various features of the collected waveforms have been described and where possible related to the composite materials internal structure. The results suggest that the glass fibre composite has the most complex anisotropy, which is due to the glass fibre composite having the more complicated internal structure of the two.

It is evident from Figure 3.11 and Figure 3.12 that propagation from transmitter to receiver leads to very complicated waveforms, containing signals from multiple wave modes (discussed further in Section 4.4.1), mode conversion and scattering. It is for these reasons, amongst others, that the neural network approach for defect location was selected.

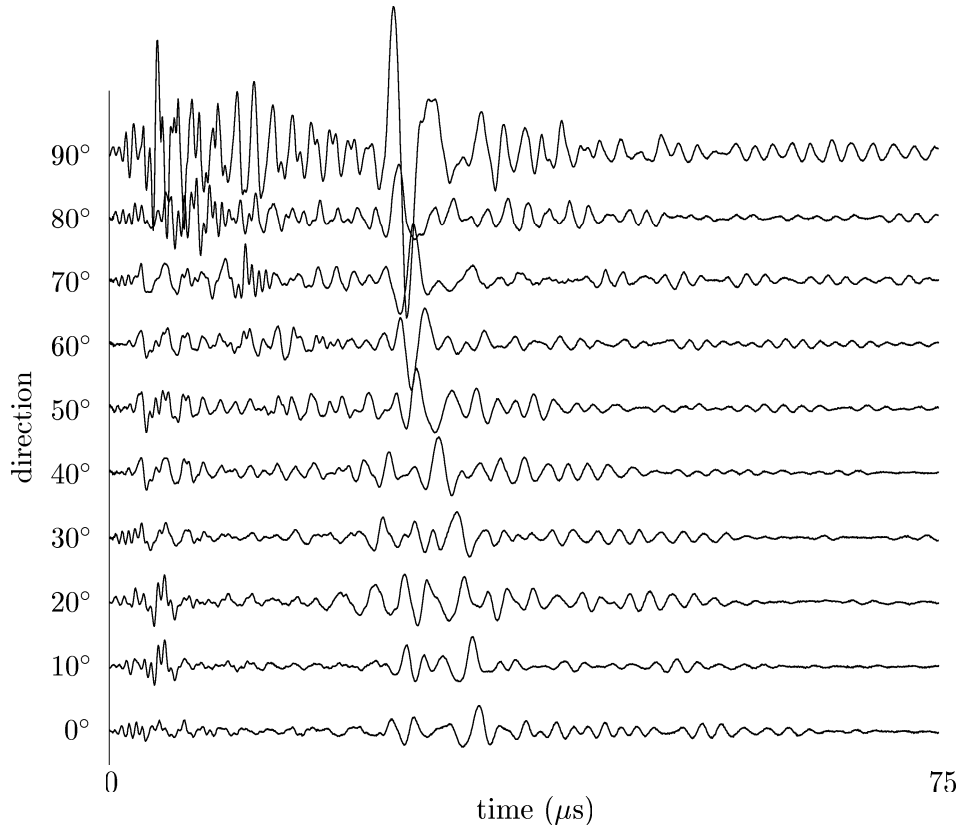


Figure 3.11: Propagation direction waveforms for the carbon material.

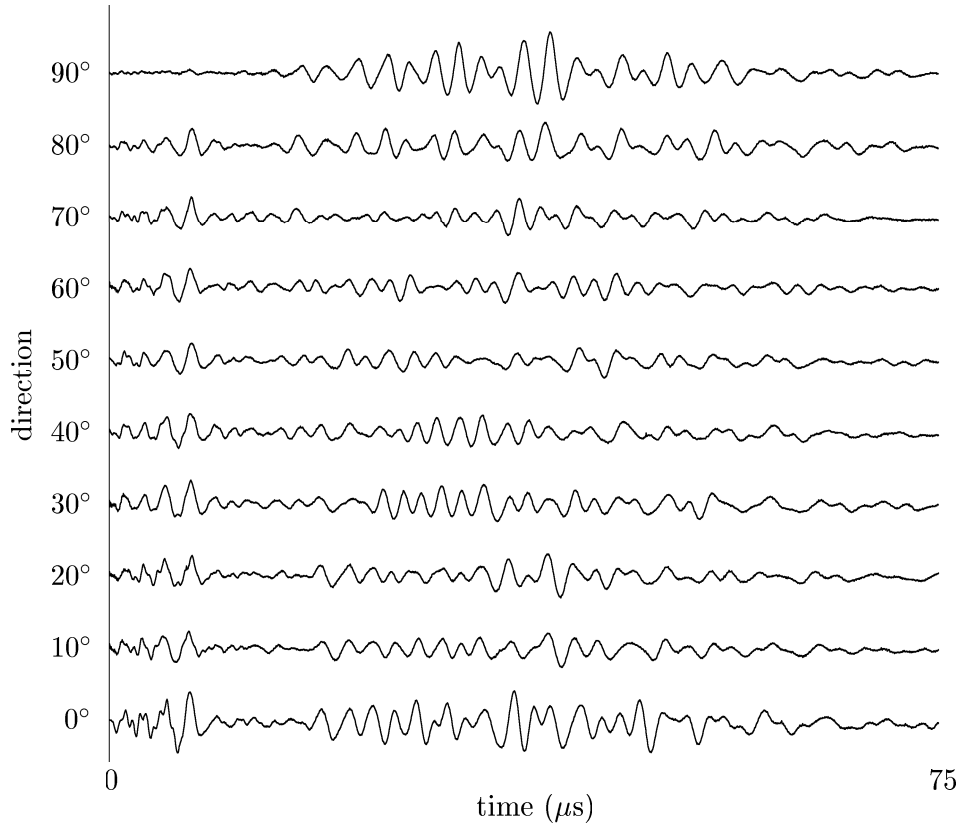


Figure 3.12: Propagation direction waveforms for the glass material.

3.6 Future Integration of the Final System

The final ultrasonic imaging system should integrate the three main components, which are; the instrumentation for data collection, data pre-processing for data reduction (details given in Chapter 4) and the neural network(s) for image generation (Chapter 5). Hence, both the neural networks and the data pre-processing were to be integrated onto the IBM personal computer platform and incorporated with the instrument control software.

All of the neural network development within this thesis utilized simulation software executing on a UNIX Sun workstation. There were two possibilities for the integration of a trained neural network onto a PC platform. The first was to convert the neural network, with its trained weights, into a sequential program replicating the neural network's input-output operation. The second method was to implement the trained neural network onto hardware compatible with a PC, either a DSP accelerator card or dedicated neural network hardware. Both methods were considered applicable.

The specifications of various dedicated neural network hardware were investigated, including the Intel 80170NX chip, Neural Technologies NT5000 system, and Neural Semiconductor SU3232 and NU32 chips. Unfortunately they all suffered from placing a limit upon the number of neurons allowed, which was generally less than the anticipated number of neurons required for the final system.

Although using a DSP accelerator card may be slightly slower than dedicated neural network hardware, it does give maximum flexibility without limiting the number of neurons. An additional advantage of using a DSP card is that the data pre-processing, which is computationally intensive, can be implemented into hardware increasing the overall system performance.

Hence, a QPC/C40ED DSP accelerator card from LSI[5] was chosen to be used to implement both the data pre-processing technique and the final neural network in a final totally-integrated system. By using the NeuralWorks simulator, a trained neural network can be converted into C code and ported onto the DSP card via its parallel C compiler. This is not achieved here, but will be the subject for further work by

[5]Loughborough Sound Images Ltd.

other researchers. In this thesis, the instrumentation described in Section 3.3.1 and Section 3.3.2.2 will be used, together with neural networks implemented on a UNIX Sun workstation.

3.7 Summary

This chapter has detailed the development of an instrumentation system used to collect ultrasonic tomography data. The initial instrumentation system was described which was used to collect preliminary data. This was followed by modifications made to the sensor array and the instrumentation system allowed an increase in the number of ultrasonic raypaths per tomographic scan. The instrument control software, which automates the data collection process and has allowed a significant increase in the amount of training and test data available for neural network development was then detailed. A detailed description of the various composite materials used for data collection was given together with analysis of the materials' ultrasonic behaviour. The method favoured for the future system integration of the instrument control software with the neural networks and data pre-processing techniques was also described.

Chapter 4

Data Acquisition and Pre-processing Techniques

4.1 Introduction

The instrumentation systems described in Chapter 3 have been shown to be suitable for the collection of data in the form of ultrasonic waveforms for pre-determined paths across an object. These are then to be used for imaging samples containing defects, and the aim of the present work is to use a neural network approach for this purpose. However, before this can be done, various factors have to be considered.

The first of these is how to train the neural network using defects, i.e. which defects should be chosen, and how these are moved through the scanning area to allow training data to be collected. Secondly, the geometry of the sensor array will have an effect, in that the positions of the receiving pinducers will affect the resulting image, and some arrangements are likely to be more effective than others. Finally, consideration of the treatment of the waveforms before input to a neural network has to be given. Because of the large number of data points involved in a typical waveform (2000 single precision floating point numbers) and the total number of raypaths (up to 232 with the 40 transducer configuration), it would not be possible to use this data directly for input to a neural network. Thus, some form of pre-processing is necessary, to select features to which the neural network will be sensitive.

In the following, these three factors are discussed. In Section 4.2, the types of defect and their location on an imaginary grid within the sensor array are discussed. Section 4.3 presents simulation analysis on the effect of receiver location around the array, and a recommendation given of the optimum geometrical arrangement for scanning. Then, Section 4.4 describes the methods of data pre-processing that were considered, including time and frequency techniques. Finally, this is followed by a comparison of the pre-processing techniques using multi-variant cluster analysis.

4.2 Defects and their Locations

Experiments were performed with a number of flat sheet samples of approximate size 300mm by 300mm. The thickness of the samples varied according to the material. The glass fibre reinforced composite samples were either 6.5mm thick for type GFRC(a) or 3.5mm thick for type GFRC(b). The carbon fibre reinforced composite samples, CFRC, were 2mm thick. Three main types of defect were used throughout this work with each type being presented to one of the three different composite materials. Three defect types are designed to produce similar acoustic properties to that of a real delamination and may be labelled as 'artificial', 'simulated' and 'induced'.

To be used mainly for the collection of training and testing data, 'artificial' defects were introduced to the GFRC(a) samples. The artificial defect took the form of a circular hole which was drilled through the thickness of the sample at the centre. Each sample of the GFRC(a) composite had a differently sized defect or hole, with diameters of 20mm, 10mm, 5mm and 1mm all being used.

The 'simulated' defects were introduced during the manufacture process and utilised the CFRC material. Essentially, square patches of some of the middle layers of the laminate were replaced by other materials. Both Teflon and chopped fibre patches were used. The Teflon defects consisted of 8 plies thick squares of varying size, lengths of 6mm, 12mm, 25mm and 50mm were used. The chopped fibre defects used the same size of square, 25mm length, but of varying thickness, including 2, 6, 8 and 12 plies. Two CFRC samples were used, one containing the four different teflon defects and the other the chopped fibre defects.

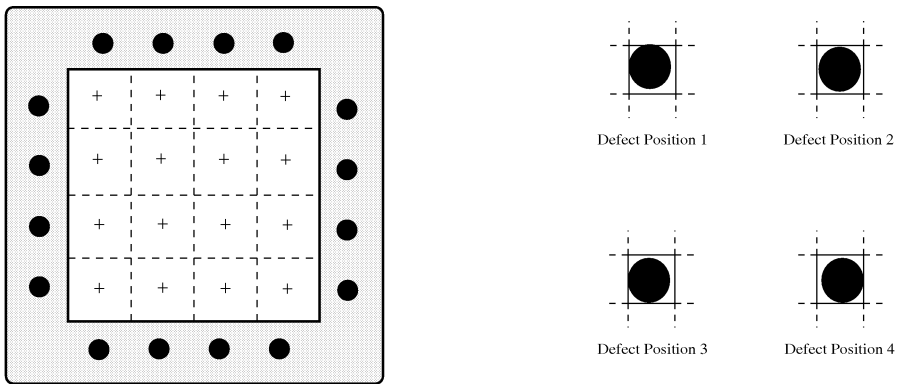
Lastly the 'induced' defects took the form of controlled impact damage of known energies. Samples of the GFRC(b) material were struck with a metal ball. The energies supplied by the impacts were recorded and include, 18J, 16J, 14J, 12J, 10J, 8J and 6J.

Two different methods have been employed for the systematic location of defects throughout the scanning area for the collection of training and testing data. Both involve the segmentation of the scanning area by an imaginary grid. Initially, the scanning area is divided into a four by four imaginary grid, as shown by Figure 4.1(a), with each grid referencing a unique 20mm by 20mm portion of the scan area. For each of the sixteen grids, the defect is located at the centre of the grid. During data collection examples are collected by offsetting the defect's centre with the centre of the grid as shown in Figure 4.1(b) to give four exemplars at each grid location. The second method is used for the collection of training and testing data from the 40 transducer sensor array. The original sixteen grids are again divided into sixteen regular squares each representing a unique 5mm by 5mm portion of the scan area, as shown by Figure 4.2. Again the centre of each grid determines the location of the centre of a defect, however there are no offset positions as before.

The location of defects for the collection of validation data takes several forms. Typically validation data will consist of examples of defects positioned either in the central four defect locations, labelled 6, 7, 10 and 11 as given by Figure 4.3, or the nine locations shown in Figure 4.4, with the crosses indicating the placement of the defect's centre. When the size of the validation defects are unknown or exceed the size of one defect location, as defined by the four by four grid, the validation defects will be positioned either in the centre of the scanning area, that is within defect locations 6, 7, 10 and 11, or in the corners of the scanning area, top left corner is defined by locations 1, 2, 5 and 6, bottom right corner by locations 11, 12, 15 and 16, etc.

Additional validation data used a prototype demonstrator Jaguar wing[1] made from cross ply carbon fibre. The wing is approximately 5.5m in length with the width varying from 0.45m at the tip to 1.33m along the fuselage side, giving an approximate area of 4.9m^2. The carbon fibre skin varied in thickness from 4mm to 10mm. Figure 4.5

[1]The Jaguar wing was supplied by DRA, Farnborough.



(a) Imaginary 4x4 grid. (b) Offset positions.

Figure 4.1: Defect locations for the 16 transducer configuration.

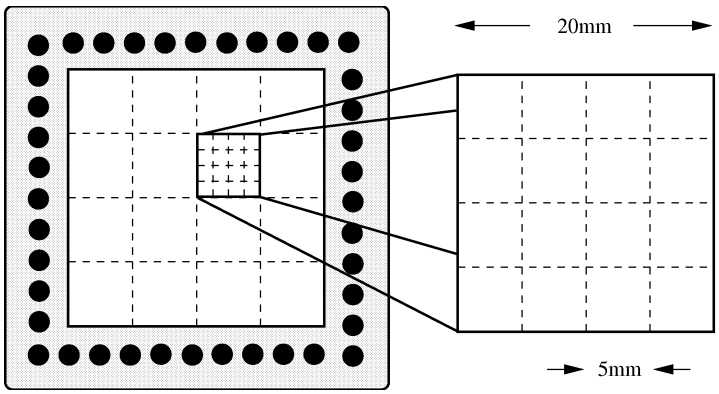


Figure 4.2: Defect locations for the 40 transducer configuration.

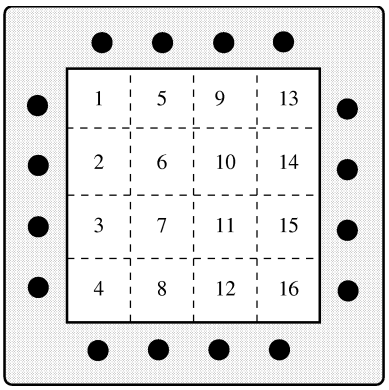


Figure 4.3: Labelling convention for defect locations.

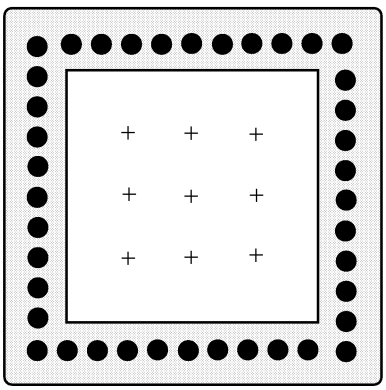


Figure 4.4: The nine validation locations for a defect.

illustrates the Jaguar wing together with the various defects it contains. Defect numbers 8 and 9 were sampled, and located within the centre of the scanning area (somewhere within locations 6, 7, 10 and 11). Both defects are impact damage, defect 8 has visible surface damage and was formed by a pointed impactor of 1.95kg mass and having 72.4J of energy, while defect 9 is known as a BVID (Barely Visible Impact Damage) defect and was formed by a 1.8kg ball having 63.6J of energy and an impact diameter of 75mm. The carbon fibre skin is 4.1mm thick at the location of both defects.

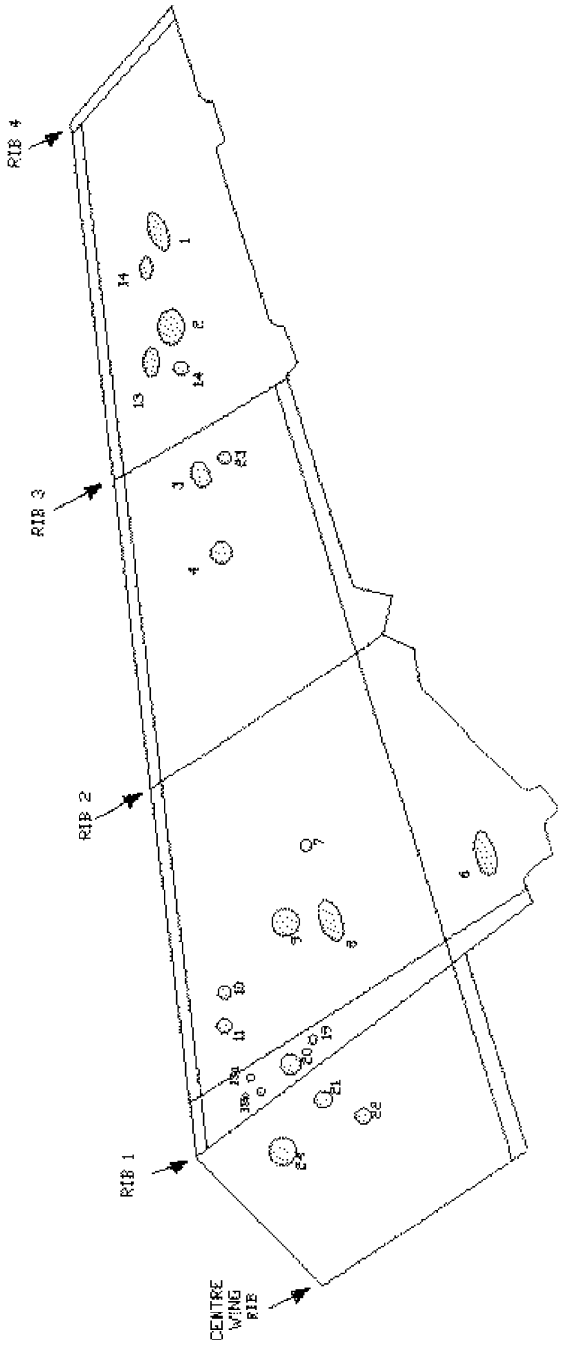


Figure 4.5: Defect locations within the Jaguar wing.

4.3 Simulating the Receiver Arrangement

The arrangement of the receivers within the sensor array alters the propagation paths of the ultrasonic waveforms collected and thus can affect the resolution abilities of the neural network. With 40 transducers, eight acting as receivers, there exists over 76.9 million possible arrangements of the transducers. Even taking into account arrangements which are symmetrically similar, there are still far too many to allow an exhaustive search to find the arrangement which optimizes the neural network performance. Despite having a semi-automated data collection method, the number of possible arrangements and the amount of data required to test each one, lead to the conclusion that it is impractical to determine the optimum receiver arrangement experimentally. Thus, simulation software has been developed, which simulates the raypaths generated by a specified receiver arrangement and produces results predicting the defect detection performance.

4.3.1 The Simulator

Simulation software has been written using the C++ programming language executing under the UNIX X-windows environment on a Sun workstation, the program code is given in appendix C. The simulator measures the relative probability of defect detection over the entire scan area for a given tomographic arrangement of ultrasonic transducers. A given simulation initiates with the specification of the arrangement of the transmitters and receivers within the sensor array, using combinations possible with 32 transmitters and 8 receivers. Then, a variety of different sized defects, ranging from 1mm to 20mm in diameter, are passed through the scan area with all raypath interceptions being recorded. The main results of the simulation are a defect detection map of the scan area, and a plot of all the raypaths within the scan area.

4.3.1.1 Simulation Assumptions

Two assumptions are required in order to make the simulations manageable and productive. Firstly, the simulator will only consider the direct path raypath between a given transmitter and receiver pair. Reflected or refracted raypaths are not simulated.

This may be justified because the experimental data collection method utilises a fixed time window which is just big enough for the direct path of the longest raypaths within the sensor array.

Conventional methods of tomographic reconstruction assume that a raypath is a line integral of the chosen parameter[2] over its path through the scan area. Increasing the number of line integrals that intercept at a given point (or pixel[3]) increases the information known about the parameter at that point, and so allows the parameter value given to that pixel by the reconstructional method to be increasingly accurate [50]. As the presence of the defect will change the state of the raypaths passing through it (e.g. by the time delay in the time domain), the more raypaths affected by the defect the greater the probability of detection and correct location. Thus, it is assumed that the more raypaths passing through a pixel in the scan area, the greater the probability of defect detection at that pixel by the neural network.

4.3.1.2 Simulator Analysis

The analysis available can be categorised into three groups: the defect detection map, the raypath plot and the raypath graph. For each category details are given of its physical relevance, together with the optimum result required for desirable neural network performance.

The defect detection map gives an indication of where in the scan area a defect is most likely to be detected. For each pixel location within the scan area, the number of raypaths intercepting defects centred at the given location is calculated. Intuitively, the requirement is for an arrangement which gives a relatively even probability of defect detection throughout the entire scan area; however, maps with small local variations are acceptable. All the maps given in Section 4.3.2 are grey scale pixel maps, the colour of each pixel corresponds to the number of raypaths intercepting defects at that pixel. The grey scale has been chosen so that it is the same for all maps, allowing easier comparisons.

[2] If time-of-flight data is experimentally collected then the parameter is acoustic slowness (the inverse of velocity) and will produce a slowness image. If amplitude data is collected then the parameter is acoustic attenuation and will produce an attenuation image.

[3] Conventional methods divide the image area into an arbitrary array of pixels, not to be confused with the neural network output image resolution

The raypath plot illustrates areas within the scan area which have few or no raypaths, thus highlighting small areas which potentially might have a larger minimum detectable defect size than the rest of the scan area. The overall view of these plots show the raypath density variation. The requirement is for an even raypath density[4] while minimizing area sizes within the scan area with no raypath coverage. The raypath plots are similarly grey scaled allowing pixels with more than one raypath interception to be highlighted, however, the main purpose of these plots is to illustrate areas with no raypath coverage.

The raypath graph shows which raypaths intercept the most (and least) number of defects in the scan area. This graph also shows the relative distances travels through the scan area by each raypath (the more defects intercepted implies larger distance travelled through scan area). This may also be interpreted as the amount of information each raypath potentially has on the presence of a defect within the scan area. A raypath graph which illustrates an even spread of information over all the raypaths will allow the development of a robust neural network, capable of dealing with problems of sensor malfunction.

4.3.2 Simulation Results

As previously mentioned, there are over 76.9 million possible receiver arrangements with a configuration having 32 transmitters and 8 receivers. Even using the simulator it would not be feasible to investigate all possible arrangements. However, consider that the arrangements group into families, for example, the raypaths generated with an arrangement that has all the receivers along the right hand side of the sensor array will form a mirror image of that with the receivers placed on the left hand side. The arrangement families consist of arrangements with receivers along one, two, three and four sides of the sensor array.

Arrangement families, having similar raypaths, will by definition, have similar features within the defect detection maps and raypath plots. Even those arrangements which are not symmetrically related will have common features.

[4] An even defect detection map infers the same number of raypath interceptions over any part of the scan area and implies even raypath density.

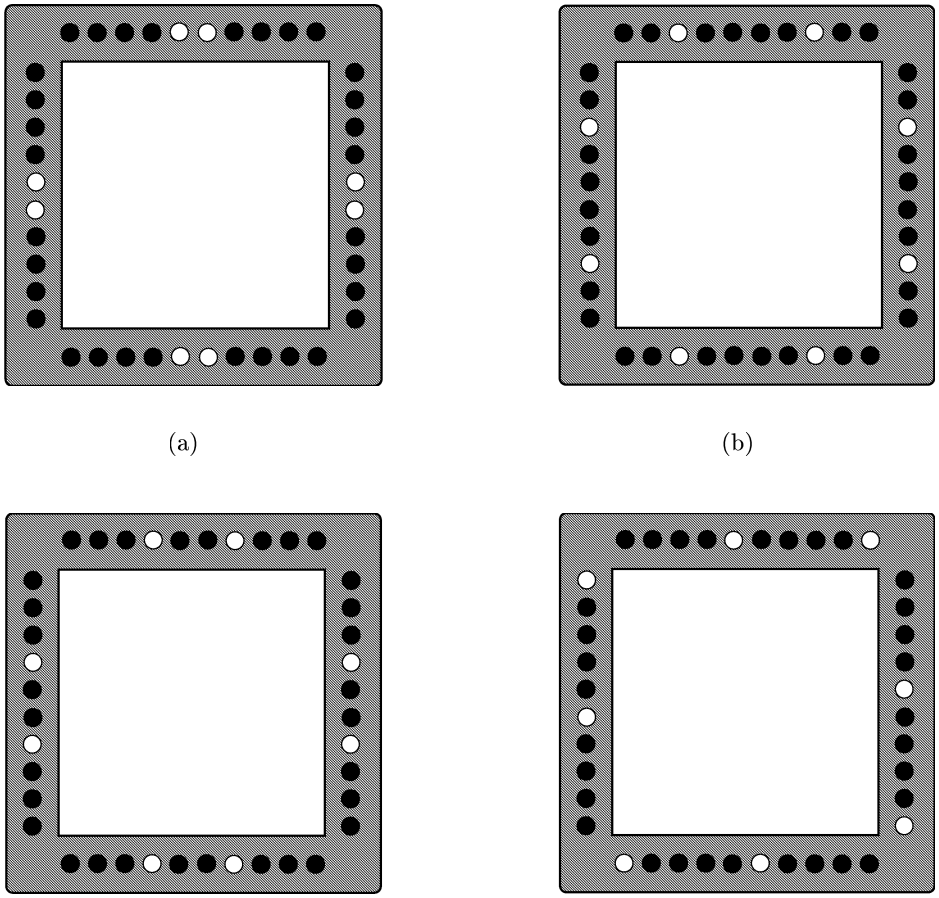
Many examples from all four arrangement families have been simulated, however, simulations with the eight receivers separated into four sets of two (one set per side) achieved the most desirable results. Thus, concentrating the simulations on this family of arrangements, and in particular, to determine the optimum grouping or separation of the receivers in the four sets, as illustrated in Figure 4.6. The defect detection maps and raypath plots for the four arrangements shown in Figure 4.6 are given in Figure 4.7 to Figure 4.10. The raypath graphs for arrangements (b) and (d) are given in Figure 4.11.

The decision criteria for a comparison of the simulation results are as follows. Inspection of the raypath plot for an even distribution of raypaths (even raypath density). Visual inspection of the defect detection map for larger areas of similar colour (implying even distribution of similar defect detection probabilities). And then a search for few, if any, areas with no raypaths (no raypath coverage). If a comparison appears to be tied at this stage, the raypath graphs should be used to determine which set of raypaths have the most evenly distributed potential information about defect detection.

The most desirable results were achieved with receivers along all four sides. The arrangements shown in Figure 4.6(b) and Figure 4.6(d) appear to have the most even defect detection maps, as shown in Figure 4.8(a) and Figure 4.10(a), and their raypath plots are very similar (Figure 4.8(b) and Figure 4.10(b)). Inspection of the raypath graphs, Figure 4.11, however, demonstrates that arrangement (b) has a more evenly distributed potential information within the raypaths.

Hence, for all the following experiments using the 40 transducer sensor array the receiver arrangement given in Figure 4.6(b) has been used.

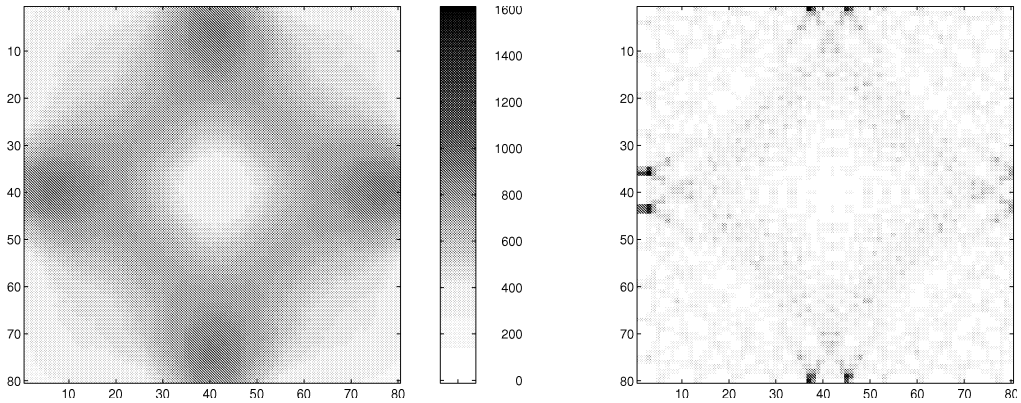
Note that there is a difference between the number of waveforms generated with the experimental arrangement compared with the simulated arrangements. This is due to the experimental arrangements having tranducers on the corners, which are effectively on two sides of the square array, and so reduce the total number of raypaths which travel through the scanning area. The assumption is, that because the simulated arrangements are used comparatively, small differences between corresponding simulated and experimental arrangements are negligible.



(a) (b)

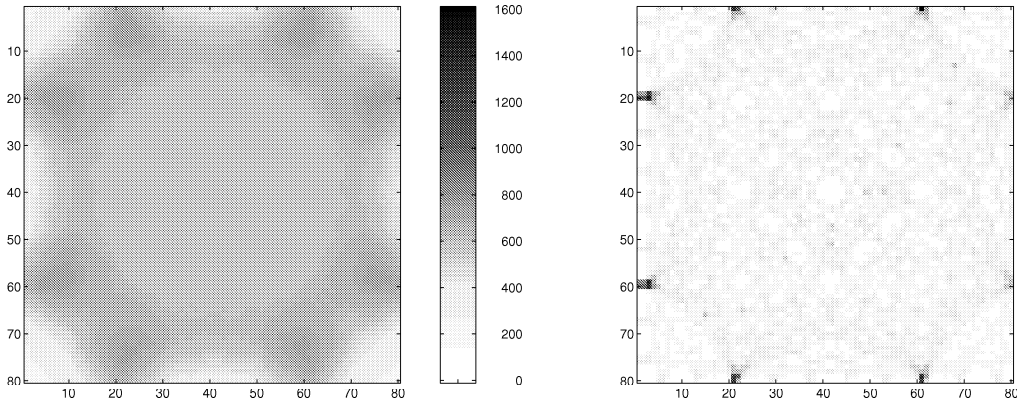
(c) (d)

Figure 4.6: Arrangements for receivers on all four sides.



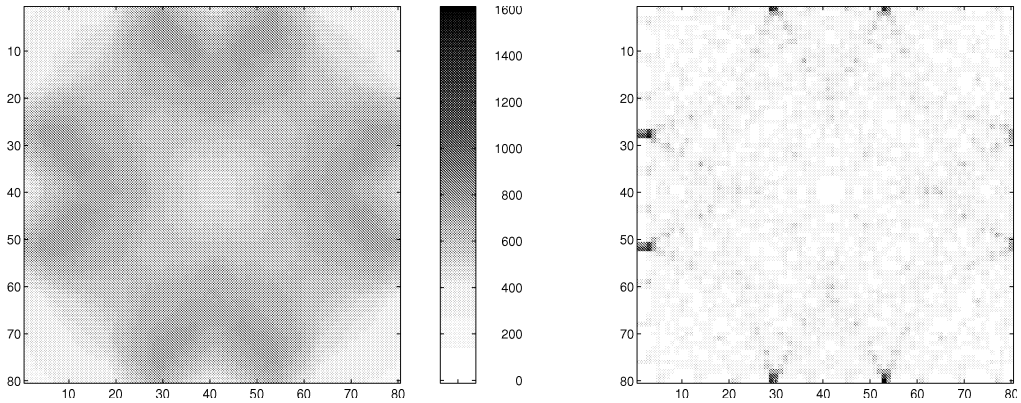
(a) Defect detection map (b) Raypath plot

Figure 4.7: Receivers on four sides, arrangement (a).



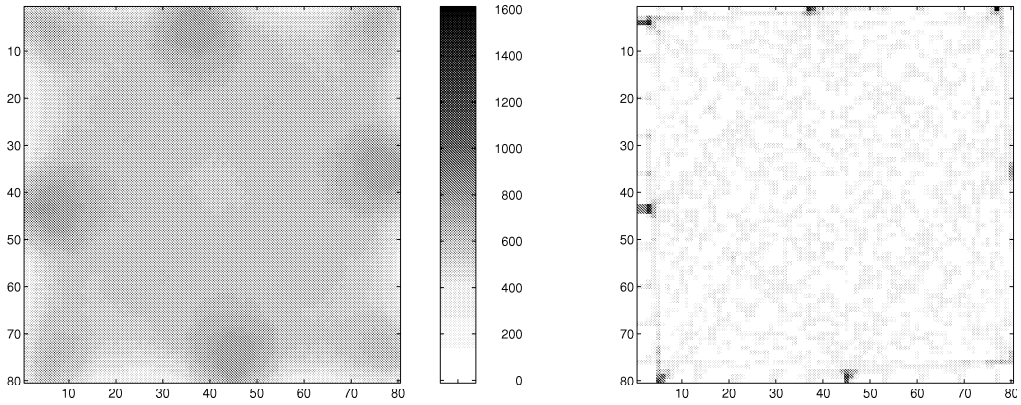
(a) Defect detection map (b) Raypath plot

Figure 4.8: Receivers on four sides, arrangement (b).



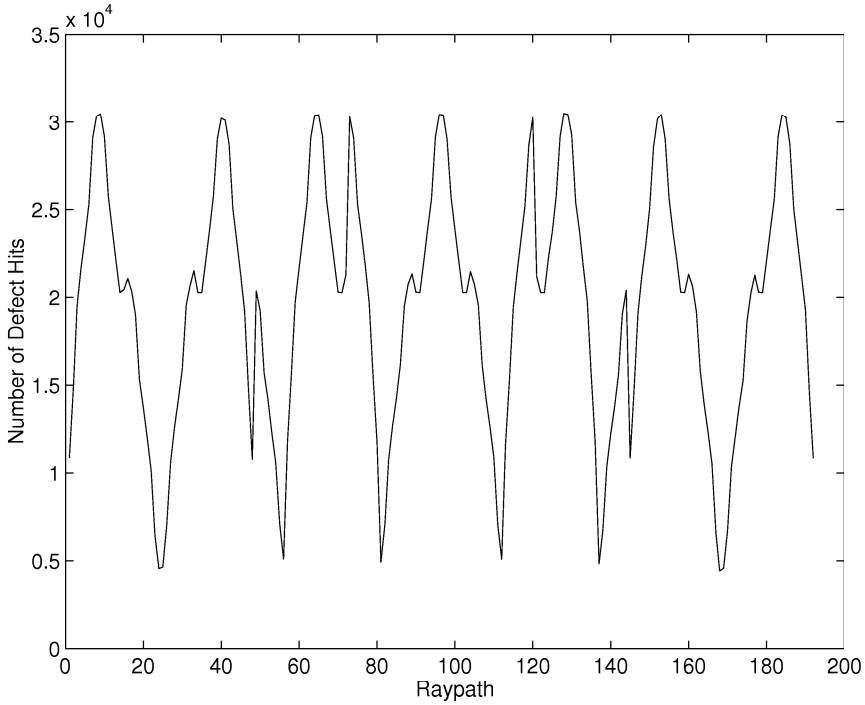
(a) Defect detection map (b) Raypath plot

Figure 4.9: Receivers on four sides, arrangement (c).

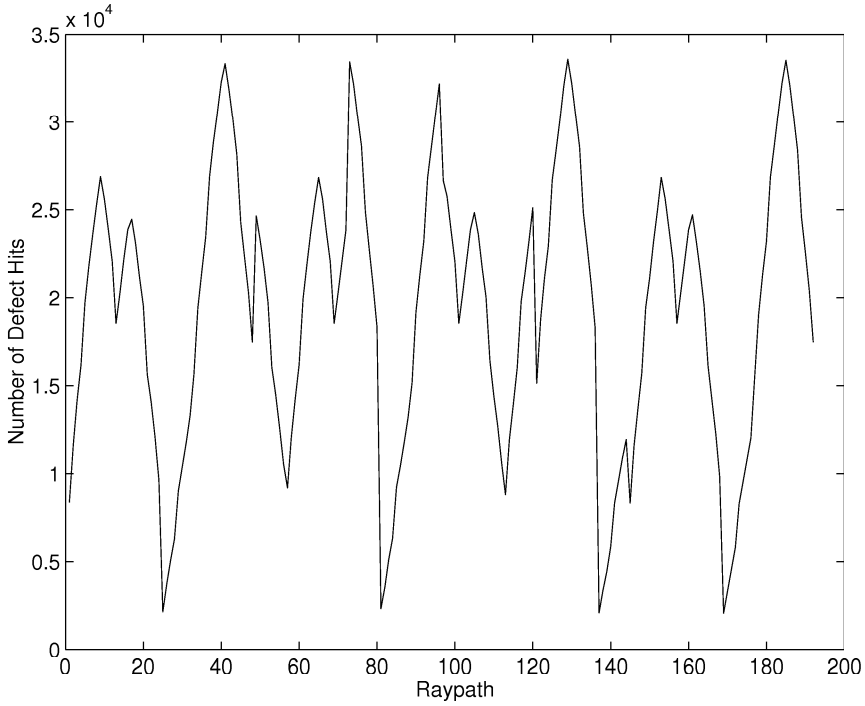


(a) Defect detection map (b) Raypath plot

Figure 4.10: Receivers on four sides, arrangement (d).



(a) Raypath graph for receivers on four sides, arrangement (b)



(b) Raypath graph for receivers on four sides, arrangement (d)

Figure 4.11: Comparison of raypath graphs.

4.4 The Data Pre-processing Techniques

A well performing pre-processing technique relies on selection of a measurement which reflects changes in features of the raw data caused by changes in the target classification. Throughout this work the main interest is in the absence or presence of a defect and as a step to determine of effective processing (raw data and processing), a comparison of ultrasonic waveforms with and without the presence of a defect could indicate possible protocols. Previous studies by other researchers have indicated that pre-processing techniques operating in both time [41] and frequency [7, 48] domains are appropriate for the classification and detection of defects using ultrasonic signals. Before selecting measures which appear, by comparison, to be affected by the presence or absence of a defect, identification of the ultrasonic modes present in the collected waveforms is performed.

All of the data pre-processing techniques detailed below were implemented in Matlab [98], a matrix manipulation and display package, executing under UNIX on a Sun workstation.

4.4.1 Identification of Ultrasonic Modes

Due to the highly anisotropic nature of the composite materials being used, no single wave mode may be assumed and it is anticipated that several modes will be evident. However, determination of the specific modes present was done empirically. A typical time-domain ultrasonic waveform is given in Figure 4.12, sampled with the GFRC(a) material, 80mm spacing between transmitter and receiver which are positions perpendicular to the uni-directional fibre bundles and with the absence of a defect. Because of the narrow frequency bandwidth of the ultrasonic pinducers, the received waveforms are not that well defined, this limits the visual identification of the ultrasonic modes present. Although, as the composite materials used are thin plate samples, with a thickness approaching the wavelengths of ultrasound present, Lamb modes are anticipated, however, the possibility of bulk and surface waves could not be discounted. Theoretical analysis allows prediction of possible ultrasonic modes but requires measurement of the velocity of the compression and shear waves within the material.

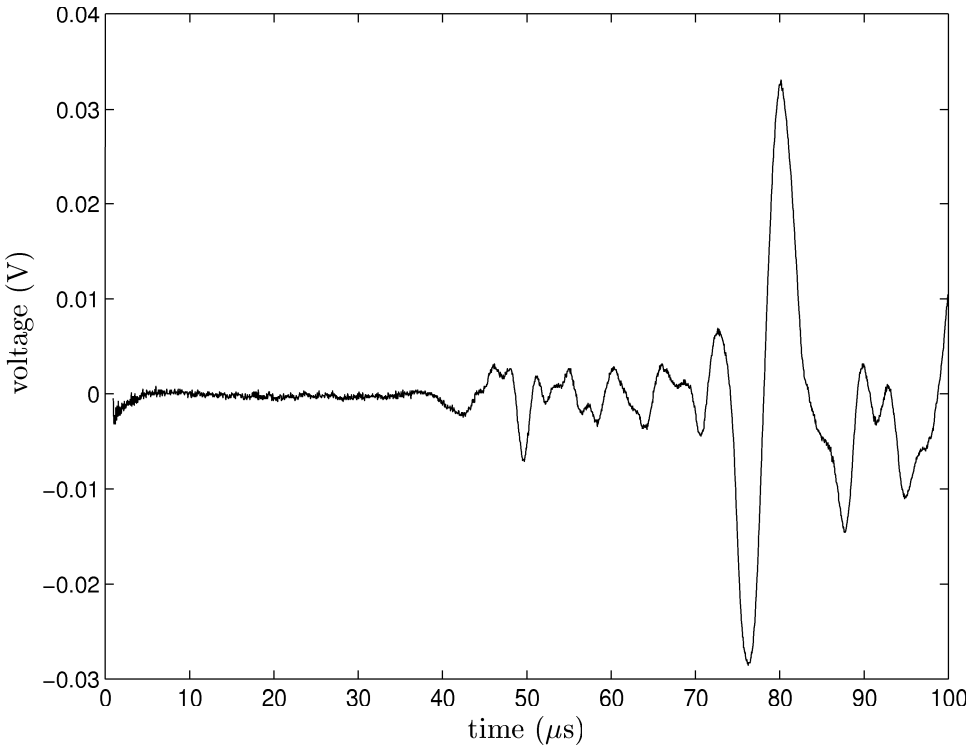


Figure 4.12: Typical time domain waveform for the GFRC material.

Due to the anisotropic nature of the GFRC(a) material both the compression and shear velocities will be dependent on the propagation direction. Four small samples of the composite material were formed and were exactly sized at 2cm length (with the fibre bundles direction), 6.5mm width (across the fibres) and 6.5mm depth (through thickness). The first time of arrival (time of flight) was measured along the three main directions, as shown by Figure 4.13, for both ultrasonic waves. The results of these measurements are given in Table 4.1. The table also shows calculated velocities for each direction and wave, with the minimum, maximum and average velocities of the four samples given.

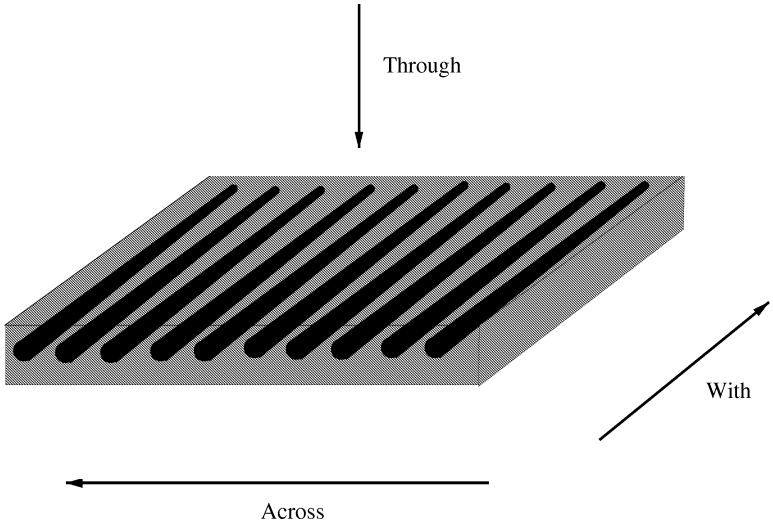


Figure 4.13: Measurement direction relative to the unidirectional fibre bundles.

Table 4.1: Measurement of ultrasound velocities in the GFRC(a) material.

	Time of Flight (μs)				Velocities (m/s)		
Sample	1	2	3	4	Min	Max	Average
Direction	Compression Wave						
With	6.02	6.30	5.92	5.88	3174.60	3401.36	3319.15
Across	2.02	2.11	2.14	2.24	2901.78	3217.82	3059.39
Through	2.74	2.76	2.72	2.74	2355.07	2389.70	2372.32
Direction	Shear Wave						
With	9.98	10.32	12.24	10.66	1633.98	2004.00	1863.03
Across	2.49	2.59	2.50	2.56	2509.65	2610.44	2564.79
Through	3.05	3.19	3.28	3.06	1981.70	2131.14	2068.66

At certain frequencies the bulk waves which form the higher order Lamb waves will have wavelengths which are the same or multiples of half the plate thickness. When this occurs the travel direction of the reflected bulk waves become normal to the surface causing the group velocity of the Lamb wave to tend to zero as its effective wavelength decreases. This is called the phenomenon of cut-off. The relationships between the plate thickness D and the bulk wave wavelengths, λ_l for longitudinal and λ_t for transverse, for the various orders of Lamb modes [99, 100] are expressed as

$$\left.\begin{array}{rcl} D & = & \frac{\lambda_l}{2}, \frac{3\lambda_l}{2}, \frac{5\lambda_l}{2}, \cdots \\ D & = & \lambda_t, 2\lambda_t, 3\lambda_t, \cdots \end{array}\right\} \text{for symmetric modes} \tag{4.1}$$

$$\left.\begin{array}{rcl} D & = & \lambda_l, 2\lambda_l, 3\lambda_l, \cdots \\ D & = & \frac{\lambda_t}{2}, \frac{3\lambda_t}{2}, \frac{5\lambda_t}{2}, \cdots \end{array}\right\} \text{for antisymmetric modes} \tag{4.2}$$

The actual cut-off frequencies for the various orders of Lamb modes may be calculated by substituting $\lambda_l = v_l/f$ and $\lambda_t = v_t/f$ into the previous equations. Where the experimentally measured values, taken from Table 4.1, for the longitudinal, v_l, and transverse, v_t, velocities are used. The calculated cut-off frequencies are shown in Table 4.2 with the average velocity values from Table 4.1 being used.

Table 4.2: Expected Lamb wave cut-off frequencies given the measured bulk wave velocity (MHz).

average velocity	symmetric Lamb modes					
	s_1	s_2	s_3	s_4	s_5	s_6
With	0.255	0.286	0.573	0.765	0.859	1.276
Across	0.235	0.394	0.706	0.789	1.176	1.183

average velocity	anti-symmetric Lamb modes					
	a_1	a_2	a_3	a_4	a_5	a_6
With	0.143	0.429	0.510	0.716	1.021	1.532
Across	0.197	0.470	0.591	0.941	0.986	1.412

By comparing the cut-off frequencies calculated from the measured compression and shear wave velocities with peaks found in the frequency domain of an ultrasonic waveform, identification of Lamb modes present should be possible. The frequency

domain is illustrated by taking the FFT of the time domain waveform, Figure 4.14 shows the FFT for the waveform previously given in Figure 4.12. There are three major peaks found in the FFT, located at frequencies just below and above 0.2MHz and just above 0.3MHz. Other less prominent peaks are located around the 0.5MHz area and at 0.6MHz and 0.65MHz. These features are suggested by the frequency values given in Table 4.2, however, identification of the modes present would require further investigation and is not necessary as a neural network technique was used.

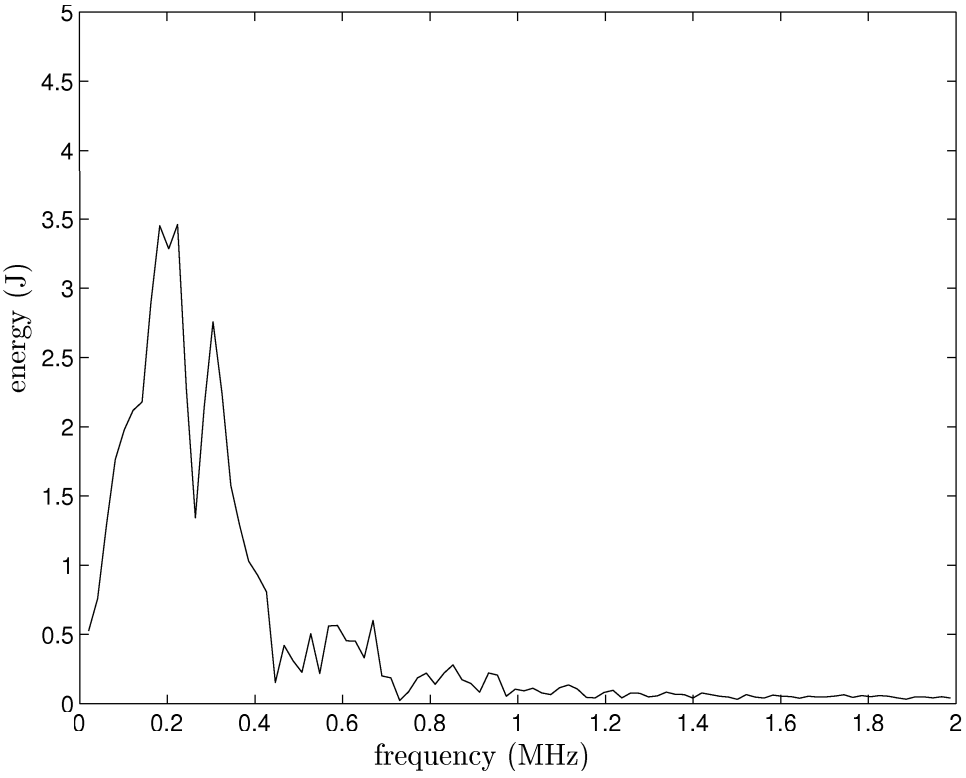


Figure 4.14: A typical frequency-domain spectrum.

4.4.2 Time Domain Measures

Figure 4.15 compares the previous time-domain waveform with an ultrasonic waveform collected with the same experimental set-up except for the introduction of a 20mm circular defect placed to intercept the raypath. The introduction of a defect has generally reduced the amplitude of the received waveform and caused a time delay in the main features. Note that only time delay measurements will be considered because the amplitude is dependent on the coupling between the pinducers and the material sample

and the life cycle of the pinducers, both of which may vary.

4.4.2.1 Time-of-Flight (TOF)

Three features in the time-domain waveforms could be easily identified both with and without the presence of a defect. Figure 4.16 illustrates the three time of flight features (A), (B) and (C). For each waveform in a tomographic scan the time values of these three features were recorded, and used to train neural networks. Time of flight pre-processing measurements (A and B) suffer from problems of accurate feature extraction and thus were rejected as a reliable data pre-processing technique.

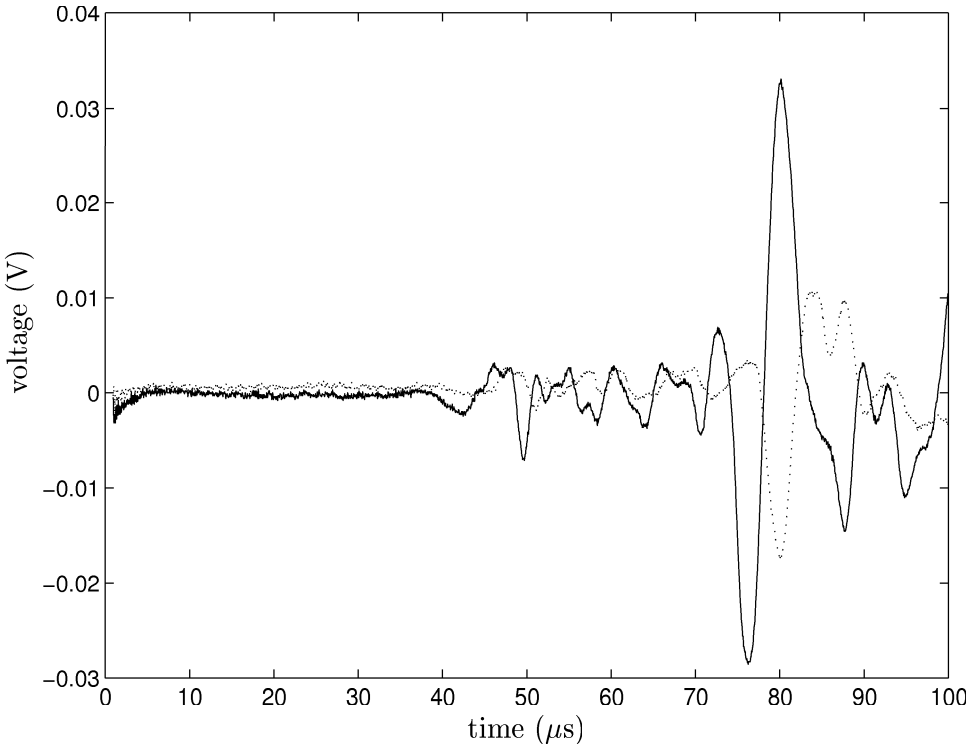
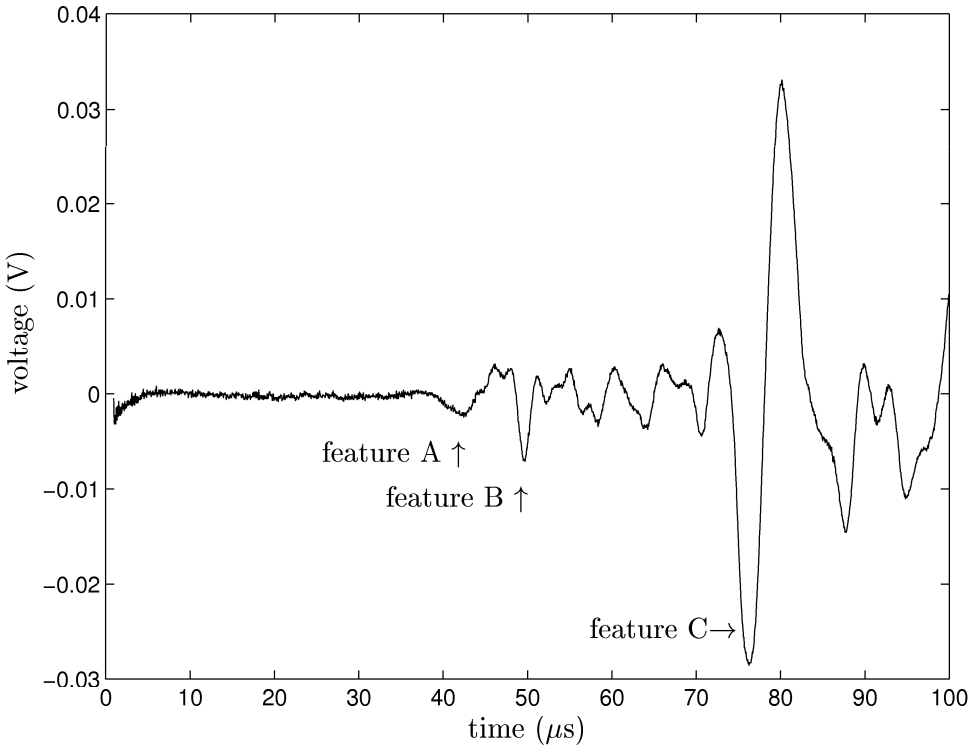
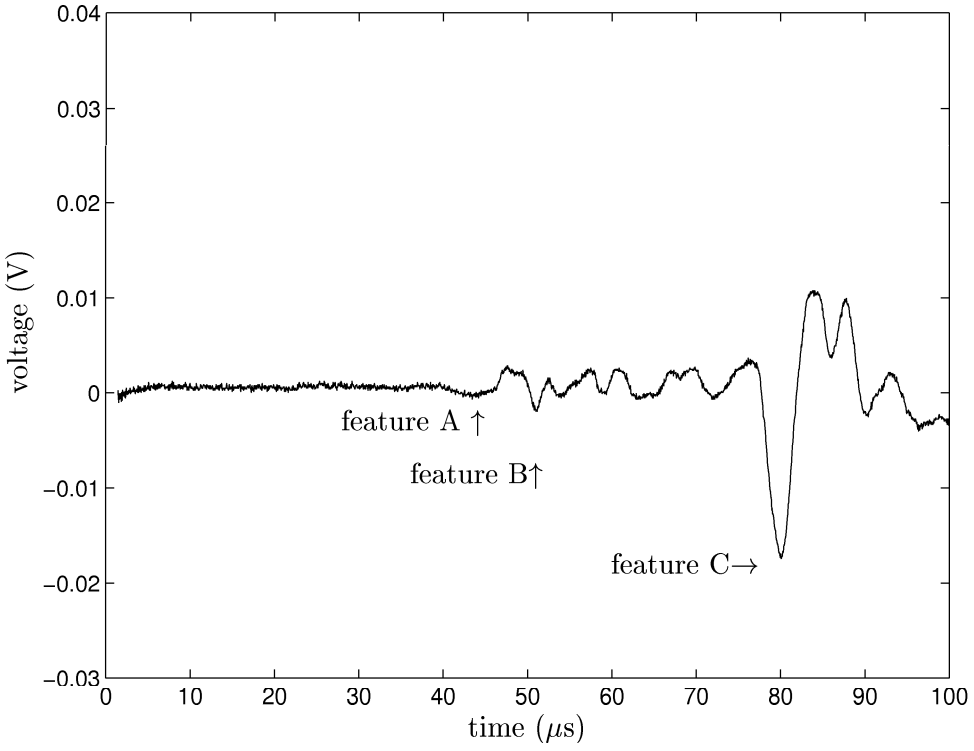


Figure 4.15: Time domain waveforms with (dotted) and without (solid) a defect.



(a) without a defect.



(b) with a 20mm circular hole defect.

Figure 4.16: Time domain waveforms with the three TOF features.

4.4.3 Frequency Domain Measures

Figure 4.17 compares the frequency-domain spectra for the same two time-domain waveforms used in Figure 4.15. In general there is a reduction of energy at most frequencies.

4.4.3.1 Centroid Value

Because of the above observation, a single value was sought which could describe the frequency content of a particular raypath. The centroid value was chosen, and is a single measurement of the frequency domain spectrum, known as the first moment of the frequency spectrum [48, 101]. It is expressed mathematically by the following expression:

$$\text{Centroid Value} = \frac{\sum_{i=0}^{n} f_i A_i}{\sum_{i=0}^{n} A_i}. \tag{4.3}$$

This equation gives the centroid value taken over a frequency range $i = 0$ to n, where n is limited by the N-point FFT such that $n \leq N$ with $N = 2048$ (the number of points in the time-domain waveform). The value of n determines the upper frequency bound of the centroid value, here the chosen value gives an upper bound of 1MHz. Hence f_i gives the frequency value at increment i and A_i refers to the amplitude of the frequency component f_i. Thus for the two spectra given in Figure 4.17 the centroid values are 348kHz with a defect and 384kHz without a defect. The centroid value will be referred to as pre-processing measurement (D).

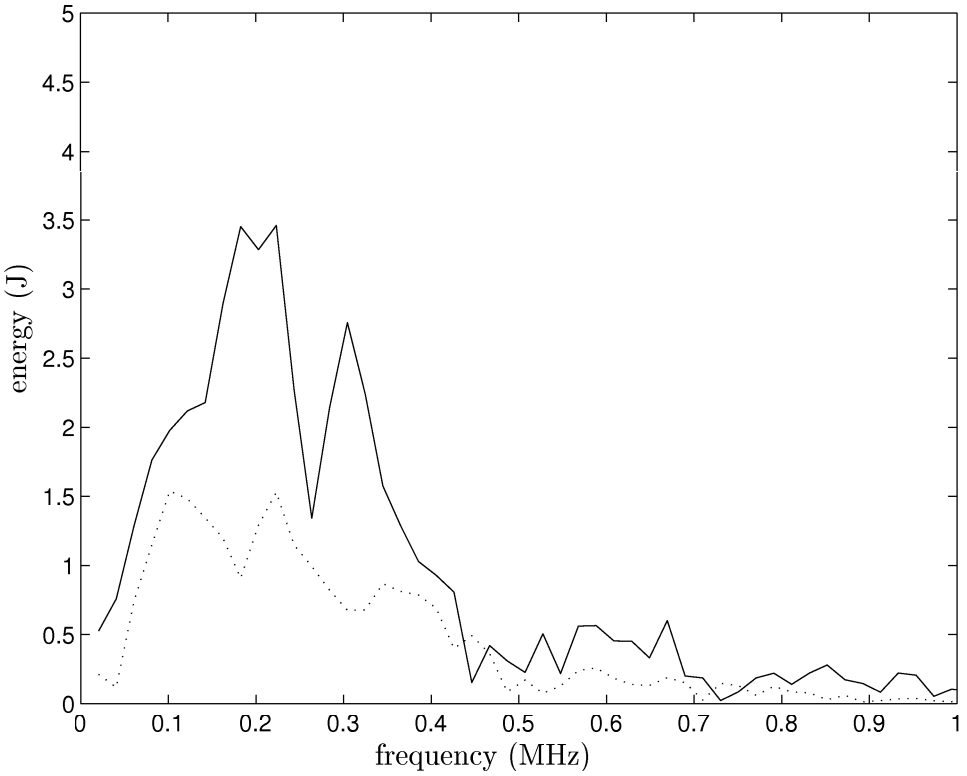
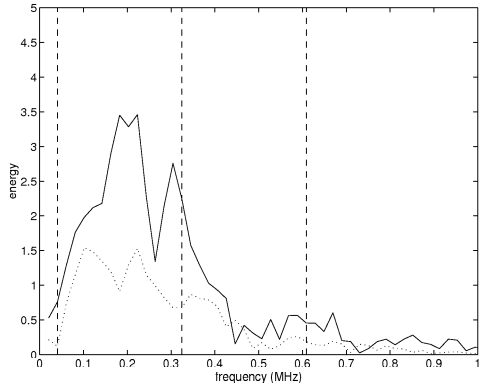


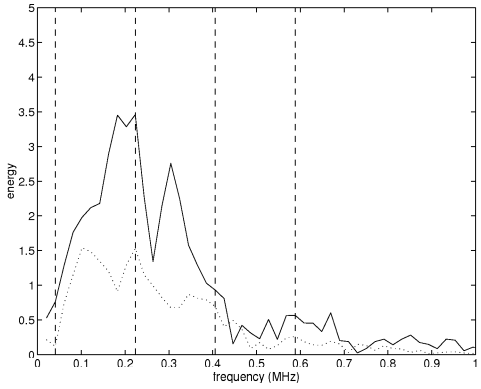
Figure 4.17: Frequency-domain spectra with (dotted) and without (solid) a defect.

4.4.3.2 Windowing

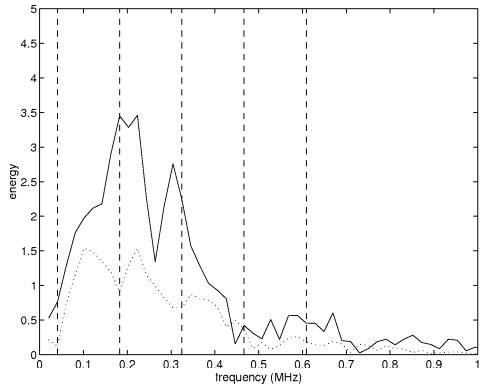
An alternative to the centroid value was to split the spectrum up into regions or windows, and determine a single value from each window. To achieve this, the frequency spectrum is divided into a number of windows as shown by Figure 4.18. The number of windows used has been varied and ranges from two to six. The frequency width of the windows depended on the number of windows being used and was configured to cover frequencies up to 0.6MHz. For each window the area under the plot was recorded. This technique is referred to as pre-processing measurements (E2) to (E6).



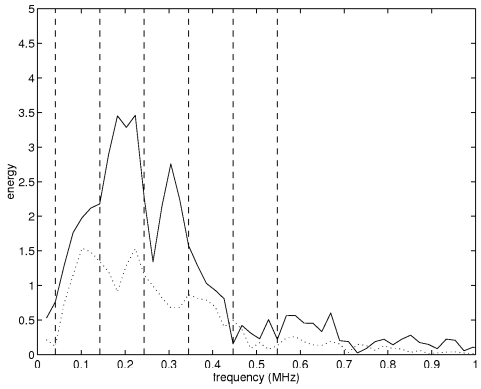
(a) two windows.



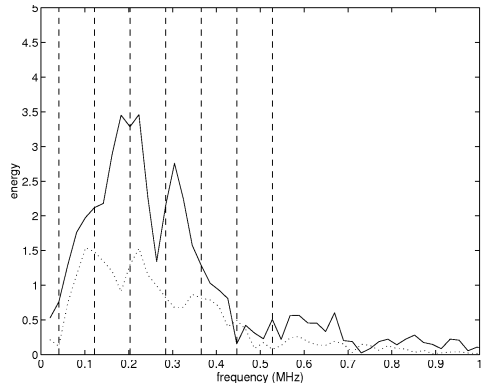
(b) three windows.



(c) four windows.



(d) five windows.



(e) six windows.

Figure 4.18: Windowing of frequency spectra.

4.4.4 Re-scaling

The re-scaling (also known as normalization) of the pre-processed tomographic data is necessary for two reasons. The first is a requirement of the neural network, detailed in Section 2.7.3, which highlights the need for the inputs to the neural network to operate within the linear portion of the activation function of the hidden layer of neurons.

The second reason is that because the "percentage change" in the time domain pre-processing measures due to the introduction of a defect will alter for different propagation raypaths. Note that the distance travelled by the ultrasonic transients varies according to the distance between the transmitter and receiver pair and that the "actual change" in the pre-processing measurements due to a defect being introduced is unrelated to the travel distance. This causes an increase in the complexity of the data. Thus, the data of each raypath measurement for examples with defects are re-scaled with respect to examples without defects producing a scaled relative change for each raypath.

4.4.5 Differencing

To allow improved neural network results with validation data, a technique called differencing is used. The aim of this technique is to limit the information about the sample materials' anisotropic nature both learnt by the neural network during training and presented to the neural network during validation. This is achieved by subtracting, for each measurement of each raypath, the value of a 'no defect' example from all examples within a data set. The differencing technique is applied after the time or frequency domain pre-processing but before re-scaling.

4.5 The Tomographic Databases

The tomographic data collected can be easily divided into two categories which relate to the sensor array. Table 4.3 describes the tomographic data collected with the two arrangements of the 16 transducer configuration, using the initial instrumentation, while Table 4.4 details the tomographic data collected with the 40 transducer configuration using the arrangement recommended from Section 4.3.2.

These databases comprise of a series of tomographic scans each consisting of a number of received ultrasonic waveforms. A given database will contain a series of tomographic scans, of a particular defect size and type, placed in different locations within the scanning area. Before neural network training and testing data may be formed, one or more of these databases is taken and pre-processed using one of the techniques previously described, re-scaled and target classifications added.

Thus, in specifying the training, testing or validation data used by a neural network, both the database and the pre-processing method used need to be indicated along with the target classification.

Table 4.3: Description of the tomographic data for the 16 transducer sensor array.

Name	Description
Tomo-1	The 16 transducer configuration, arrangement A (Figure 3.7(a)) using the GFRC(a) material with a 20mm artificial defect, placed at the 16 locations shown in Figure 4.1(a) with offset positions 1, 2, 3 and 4 shown in Figure 4.1(b). In addition 4 examples of a no defect sample are also collected giving a database of 68 patterns with each pattern consisting of 48 waveforms.
Tomo-2	The 16 transducer configuration, arrangement B (Figure 3.7(b)) using the GFRC(a) material with a 20mm artificial defect, placed at the 16 location shown in Figure 4.1(a) with offset positions 1, 2, 3 and 4 shown in Figure 4.1(b). In addition 4 examples of a no defect sample are also collected giving a database of 68 patterns with each pattern consisting of 36 waveforms.
Tomo-3	The 16 transducer configuration, arrangement A using the GFRC(a) material with a 5mm artificial defect, placed at the 16 locations shown in Figure 4.1(a) with offset positions 2 and 4 shown in Figure 4.1(b). In addition 2 examples of a no defect sample are also collected giving a database of 34 patterns with 48 waveforms per pattern.
Tomo-4	The 16 transducer configuration, arrangement A using the GFRC(a) material with a 1mm artificial defect, placed at the 16 locations shown in Figure 4.1(a) with offset positions 2 and 4 shown in Figure 4.1(b). In addition 2 examples of a no defect sample are also collected giving a database with 34 patterns, again with 48 waveforms per pattern.

(Continued on next page)

Table 4.3: (continued)

Name	Description
Tomo-5	The 16 transducer configuration, arrangement A using the GFRC(a) material with a 10mm artificial defect, placed as defect locations 6, 7, 10 and 11 (labelling shown in Figure 4.3. In addition 2 examples of a no defect sample are also collected giving a database with 6 patterns, again with 48 waveforms per pattern.
Tomo-6	The 16 transducer configuration, arrangement A using the CFRC material with the teflon and chopped fibre inclusions. With one example of each of the four Teflon defects and one example of each of the four chopped fibre defects. The two sets of four defects are positioned so that one of the four defects is located in one of the four corners of the scanning area. In addition 2 examples of a no defect sample are also collected. This is repeated so that the carbon fibres within the samples are arranged to be both aligned with and perpendicular to the uni-directional fibres of the glass fibre samples used for training data (tomo-1 to tomo-5) with respect to the sensor array to give a database with 20 patterns, each pattern consisting of 48 waveforms.

Table 4.4: Description of the tomographic data for the 40 transducer sensor array.

Name	Description
Tomo-7	The 40 transducer configuration, arrangement shown in Figure 4.6(b) using the GFRC(a) material with a 5mm artificial defect, located at the 256 positions shown in Figure 4.2. Four examples of each location were collected plus 4 examples of a no defect sample giving a data set with 1028 patterns, each patterns consisting of 184 waveforms.
Tomo-8	The 40 transducer configuration, arrangement shown in Figure 4.6(b) using the GFRC(a) material with a 10mm and 20mm artificial defect, located at the nine validation positions shown in Figure 4.4.
Tomo-9	The 40 transducer configuration, arrangement shown in Figure 4.6(b) using the CFRC material with the chopped fibre and teflon defects. One example of each of the eight defects (four chopped fibre and four teflon) plus two examples of no defect. The defects are located within the centre of the scanning area.
Tomo-10	The 40 transducer configuration, arrangement shown in Figure 4.6(b) using the GFRC(b) material with the impact damage defects, energies of 6J, 8J, 10J, 12J, 14J.
Tomo-11	The 40 transducer configuration, arrangement shown in Figure 4.6(b) using the Jaguar wing. Two examples of defects 8 and 9 (shown in Figure 4.5) and two examples with no defect.

4.6 Comparison of the Data Pre-processing Techniques

Given the various pre-processing techniques, previously detailed, a method is needed to determine the most effective technique, in terms of minimum loss of information relating to defect location, which will lead to maximum neural network performance.

Cluster analysis allows input data to be grouped into clusters of similar patterns by means of an input space distance measure. Clustering may be considered to be an unsupervised learning paradigm, requiring no user defined target classification. However, to permit a comparison of the pre-processing techniques the actual defect locations (targets) are attached to the clustered data to identify if the clusters formed relate to defect location. The optimum pre-processing technique will produce clusters that closely relate to defect location.

For the purpose of identifying clusters of similar defect locations, the 4 by 4 imaginary grid is labelled as shown by Figure 4.19. Thus, all the examples of a data set will be given the label relating to the location of the defect within the imaginary grid. Examples which have no defect present will have a zero label.

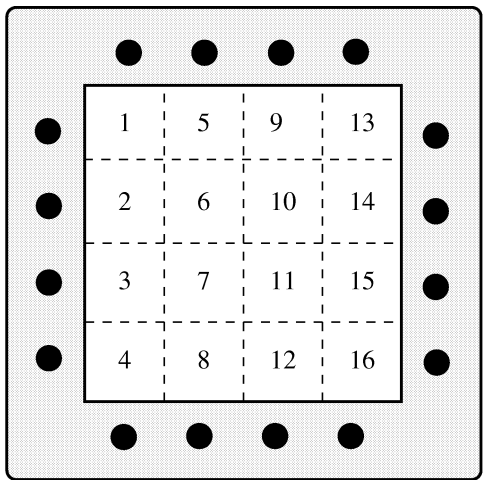


Figure 4.19: Labelling convention for defect locations.

Both linear and non-linear cluster analysis have been performed. Section 4.6.1 details the method of linear clustering, called principal component analysis, while Section 4.6.2 describes the SOM neural networks used for the non-linear clustering.

4.6.1 Principal Component Analysis

Principal component analysis (PCA) is a multivariate data analysis technique which was originally devised by Pearson [102]. The method is described by Jolliffe [103] as follows:

> "PCA is a technique of multivariate analysis used to reduce the dimensionality of a large number of interrelated variables while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables called the principal components (PCs) which are uncorrelated, and which are ordered so the first few retain most of the variation present in all of the original variables."

The mathematical definition and derivation of the principal components are given in appendix D. The PCA was implemented using Matlab [98], a mathematics matrix manipulation and display tool, executed under the UNIX environment on a Sun workstation. The PCs may be displayed in a number of ways, the simplest involves taking a plot of the first against second PC, third against fourth PC and so on.

The first eight PCs have been examined for all of the different pre-processing measures formed from databases tomo-1 and tomo-2 (20mm defect, arrangements A and B). Figure 4.20 to Figure 4.22 give the first four principal components for three of the pre-processing measures. All of the pre-processing measures, for both arrangements gave a minimum of 50% of the original data variation with four PCs and required ten PCs to give over 80% variation. Within each of the first and second PC plots (PC1 against PC2) there is good evidence of clusters which relate to input patterns of similar defect locations. However the third and fourth PC plots illustrate that the frequency domain centroid value, pre-processing measure (D), forms the most separated clusters of similar defect locations. This would suggest that despite there being little overall difference between the pre-processing techniques, measure (D) appears to have the most linearly separable clusters which relate to defect location. Hence, measurement (D) should produce good performance with a neural network.

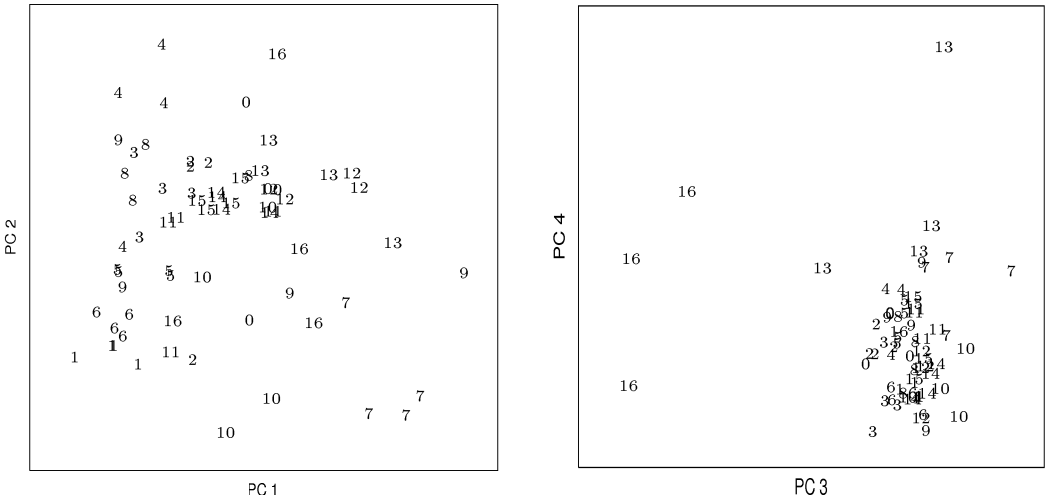


Figure 4.20: PCA for arrangement A, pre-processing measure (C).

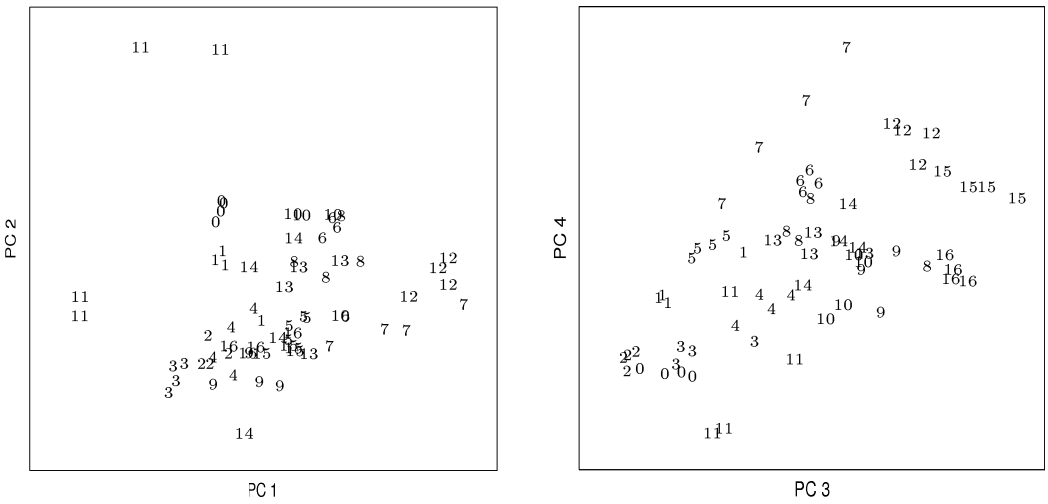


Figure 4.21: PCA for arrangement B, pre-processing measure (D).

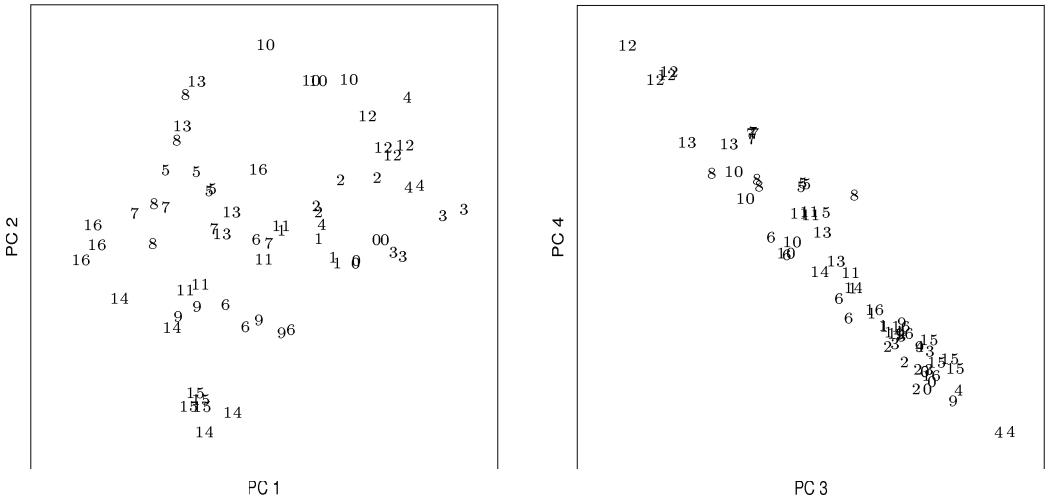


Figure 4.22: PCA for arrangement B, pre-processing measure (E2).

4.6.2 Self Organizing Maps

Three different sized SOMs were used for all the pre-processing measures and with both arrangements (databases tomo-1 and tomo-2). Specifically maps of size 10 by 10, 15 by 15 and 20 by 20 outputs have been used, however only results from the 20 by 20 maps are given. Figure 4.23 to Figure 4.25 shows the resulting maps from three SOM neural networks. A square neighbourhood function was chosen and had an initial width, μ_o, of 7 which decreased to 1. The initial learning rate, η_o, was 0.06. The maps were wrapped both horizontally and vertically and the training was stopped after 30 epochs (2040 cycles).

Visually comparing the maps, illustrates that while all of the pre-processing measures produce very good clustering of the different defect locations, the frequency domain centroid value, measurement (D), gives the best separation of clusters.

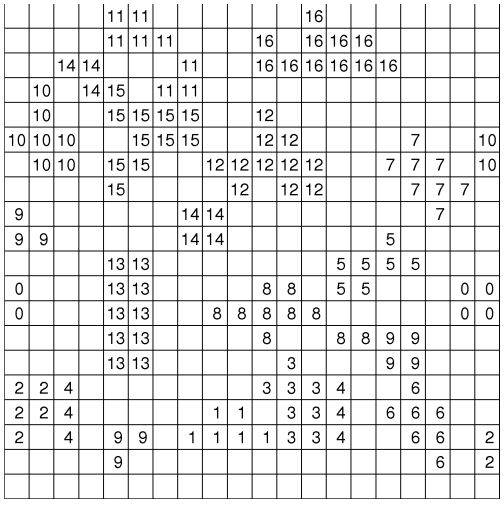


Figure 4.23: SOM for arrangement A, pre-processing measure (C).

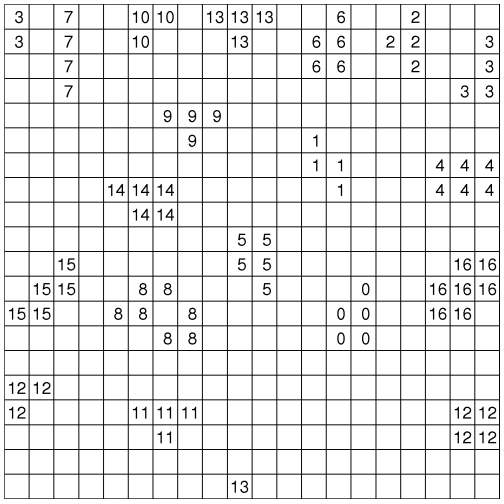


Figure 4.24: SOM for arrangement B, pre-processing measure (D).

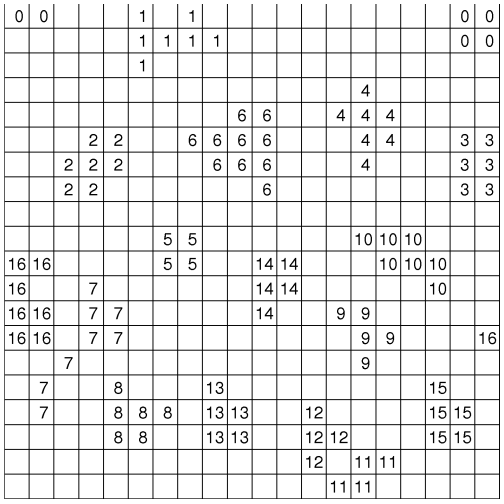


Figure 4.25: SOM for arrangement B, pre-processing measure (E2).

4.7 Summary

This chapter has shown the procedure adopted for the systematic location of defects throughout the scanning area for the collection of neural network training data. A method of simulating the ultrasonic raypaths generated with a specific receiver arrangement has been described and used to determine an optimal arrangement for the 40 transducer configuration. The various data pre-processing methods, used to reduce the dimensionality of the waveforms generated by a tomographic scan, before presentation to a neural network, have been described. The two techniques of multi-variant data analysis, used to perform an initial comparison of the pre-processing techniques, have shown no significant difference between the various pre-processing measurements, although the frequency domain centroid value, measure (D), appears to have a slight performance advantage in terms of the separability of the clusters relating to defect locations.

Chapter 5

Neural Networks for Ultrasonic Tomographic Imaging

5.1 Introduction

The initial comparison of the pre-processing techniques, using principal component analysis, has shown that the tomographic data has inherent clusters which appear to relate to the defect location within the scanning area. This is strengthened by the results of the SOM neural networks, which also found clusters relating to defect locations. This would suggest that a supervised neural network, such as standard BP, would be able to classify according to defect location, thus producing an image of the scanning area as its output.

One of the main considerations of this work is the resolution limitations of the system. There are two contributors to this limitation, the first coming from the physical bounds of the tomography, defined by the sensor array, and the second is less obvious and concerns possible limitations of the neural network system in generating high resolution images as output.

This chapter begins with an initial analysis of various data sets, placing emphasis on the investigation of complexity. This is followed by outlining the procedures adopted for neural network optimization. A detailed account of the optimization for the neural networks using data sets from the 16 transducer sensor array is given to illustrate the

methodology, while a summary of the optimization procedure is given for other data sets. Initial investigations of the performance of a neural network using the tomographic data were conducted with the data sets collected from the 16 transducer sensor array. For these experiments emphasis was placed on understanding the various data sets, in particular, selection of the pre-processing technique(s) which gave optimal neural network performance. This leads to a preliminary investigation of the ability of a neural network to determine both defect size and location. This is followed by an investigation of the data collected from the 40 transducer sensor array, including a study of the maximum output resolution possible. Finally the various neural network systems are evaluated with validation data, which includes defects sampled from an aircraft wing.

5.2 Initial Data Analysis

Although the cluster analysis given in the previous chapter has indicated which pre-processing measurements generate input data that contains the greatest amount of information relating to defect location, it would be desirable to directly assess the complexity contained within the combined input and target output data.

The tomographic databases used were tomo-1 and tomo-2, that is, the 16 transducer sensor array, arrangement A and B with a 20mm diameter circular hole defect in the glass fibre reinforced composite material. Both pre-processing measures (C) and (D) were used to give two data sets which each had 68 patterns with either 48 or 36 inputs and 16 outputs. The outputs indicate the location of a defect (see Section 5.5.1) and for each pattern only one of the sixteen outputs is active (indicating the defect). Over the complete data set a given output is only active for 4 of the 68 patterns.

According to the principle of Occam's razor [104], the complexity of the data should be matched by the complexity of the technique used to model the data. The method chosen to assess the complexity of the different data sets (input and target output) is to apply several models to the data, starting with a very simple linear model and moving onto models which allow greater flexibility in the forming of a model and thus can model more complex data.

The first model used was multiple linear regression [105, 106]. In its simplest form multiple linear regression takes all of the independent input variables in order to generate one dependent output variable as shown in the following equation,

$$\hat{o}_1 = \psi_0 + \psi_1 \mathbf{z}_1 + \cdots + \psi_k \mathbf{z}_k + \cdots + \psi_K \mathbf{z}_K + \varepsilon_1, \tag{5.1}$$

where the estimated output $\hat{o}_1$ is given by a linear combination of the input data $\mathbf{z}$, with ψ representing the parameters of the model used to scale the contribution of each input variable in the estimation of the output. ε_1 gives the error between the actual output o_1 and the model's estimate $\hat{o}_1$. Linear regression minimizes the sum of squared error over all observations (or patterns) to produce the optimal values of ψ. The linear regression model applied to the tomographic data suffered from 'biasing', continually indicating a 'no defect' irrespective of the input pattern.

The second model used was a standard BP neural network with no hidden neurons. Shown to be equivalent to a logistic regression model [107]. Neural networks were trained for both receiver arrangements and for time and frequency pre-processing techniques (C) and (D), the results of which are given in Table 5.1. The results indicate that the frequency domain pre-processing technique produces data which is lower in complexity than that of the time domain technique. Also demonstrated, is how the combination of pre-processing technique and receiver arrangement affects the data complexity, with arrangement B combined with the frequency pre-processing technique (D) giving the best testing performance and hence the least complex data. The best performing pre-processing technique (D) and receiver arrangement (B) correspond with that found earlier by cluster analysis.

Table 5.1: BP neural networks with no hidden neurons.

Pre-pro. Measure	Perform. Metric	Threshold 0.2	0.3	0.4	0.5	0.6	0.7	0.8
Arrangement A								
(C)	TP	21	22	23	26	29	31	32
	TN	511	508	505	501	497	490	474
	FP	1	4	7	11	15	22	38
	FN	11	10	9	6	3	1	0
	Sens	0.66	0.69	0.72	0.81	0.91	0.97	1.00
	Spec	1.00	0.99	0.99	0.98	0.97	0.96	0.93
	PosPV	0.95	0.85	0.77	0.70	0.66	0.58	0.46
	FalAR	0.00	0.01	0.01	0.02	0.03	0.04	0.07
	% Corr	97.79	97.43	97.06	96.88	96.69	95.77	93.01
(D)	TP	21	27	29	30	32	32	32
	TN	512	511	511	510	509	505	491
	FP	0	1	1	2	3	7	21
	FN	11	5	3	2	0	0	0
	Sens	0.66	0.84	0.91	0.94	1.00	1.00	1.00
	Spec	1.00	1.00	1.00	1.00	0.99	0.99	0.96
	PosPV	1.00	0.96	0.97	0.94	0.91	0.82	0.60
	FalAR	0.00	0.00	0.00	0.00	0.01	0.01	0.04
	% Corr	97.98	98.90	99.26	99.26	99.45	98.71	96.14
Arrangement B								
(C)	TP	17	20	21	23	28	30	30
	TN	511	508	507	503	496	485	473
	FP	1	4	5	9	16	27	39
	FN	15	12	11	9	4	2	2
	Sens	0.53	0.62	0.66	0.72	0.88	0.94	0.94
	Spec	1.00	0.99	0.99	0.98	0.97	0.95	0.92
	PosPV	0.94	0.83	0.81	0.72	0.64	0.53	0.43
	FalAR	0.00	0.01	0.01	0.02	0.03	0.05	0.08
	% Corr	97.06	97.06	97.06	96.69	96.32	94.67	92.46
(D)	TP	28	30	30	31	31	31	32
	TN	512	512	512	512	512	512	507
	FP	0	0	0	0	0	0	5
	FN	4	2	2	1	1	1	0
	Sens	0.88	0.94	0.94	0.96	0.96	0.96	1.00
	Spec	1.00	1.00	1.00	1.00	1.00	1.00	0.99
	PosPV	1.00	1.00	1.00	1.00	1.00	1.00	0.86
	FalAR	0.00	0.00	0.00	0.00	0.00	0.00	0.01
	% Corr	99.26	99.63	99.63	99.82	99.82	99.82	99.08

5.3 Practical Issues and Presentation of Results

For each neural network detailed in this chapter a series of simulations were conducted. Each simulation was given a different set of initial weight values increasing the chance of obtaining a neural network which had properly converged. Selection of the number of simulations per neural network was determined by the available resources and the computational time per simulation. Given the availability of several Sun workstations and the computation time per simulation of approximately 3 hours for NNs using the 16 transducer data, five simulations per neural network could be comfortably executed over night on one workstation. Computational time per simulation for neural networks using the 40 transducer data was slightly greater at about 5 hours, however, five simulations per NN were still performed.

The presentation of results from neural network simulations takes several formats. For investigations which compare several neural networks the following procedure is used. The number of TP, TN, FP and FN are calculated for all five simulations of a given neural network, using the middle threshold value. The average TP, TN, FP and FN values are then used to calculate the average performance measures for that neural network. However, only some of these measures may be given to compare the performance of several neural networks, typically the measures given will be the sensitivity and the percentage correct. These have been chosen because of three factors, the typical ratio between defects and no defects (the training data sets typically contain a significantly larger number of 'no defect' target outputs compared with the number of 'defect' target outputs), the typical performance of neural networks trained with defect location targets and the desired qualities of the final neural network (requiring the neural network to be a good 'defect' locator, not a good 'no defect' locator).

Presentation of results from specific neural networks, typically showing the performance measures with changing threshold values, will be taken from the best performing simulation of that NN. Also any given confusion matrices and confidence maps will be produced from the best performing simulation using the middle threshold.

5.4 Neural Network Optimization

As inferred from Chapter 2, Section 2.7 and Section 2.8, initial neural network experiments with a given data set involve the optimization of the training parameters such as learning rate and momentum and the selection of an appropriate number of hidden neurons. The methodology for neural network optimization, used throughout this thesis, is detailed below, for each of the parameters which need optimizing. This is followed by details of the optimization for the data sets generated from the 16 transducer sensor array and a summary of the optimization for the 40 transducer data.

For each given training data set a selection of different learning rates and momentums are used to train a number of neural networks. Typically the learning rate values selected will include 0.0005, 0.001, 0.005, 0.01, 0.05 and 0.1, while the momentum values selected will include 0.5, 0.6, 0.7, 0.8 and 0.9. The number of epochs to train all the different neural networks is chosen so that the neural network with the smallest learning rate and momentum can reach its trained region as defined by Figure 2.7.

Having trained all of the different neural networks, a plot of SMSE for the testing data over the training period, for each NN can be formed, and the comparison of these plots allow the optimum learning rate and momentum to be decided. However, selection of the appropriate values for the learning rate and momentum is somewhat discretionary, typically there is no obvious choice for the optimum value. The method adopted is to select the error plot which gives an initial steep decent in error from a medium starting error value followed by a continued decrease in error as training progresses and finishing at a relatively small error value.

To determine the optimum number of hidden neurons a number of neural networks will be trained using the same training and testing data sets and the same optimized learning rate and momentum values. The architecture of the neural networks will have the same number of input and output neurons, however, each NN will have a different number of hidden neurons. The range of values used for the heuristic search will typically exceed the range suggested by the application of equation 2.45. The selection of the optimum architecture may also be somewhat discretionary, in general the criterion is to maximize neural network testing performance while limiting the

number of hidden neurons used. Given the optimum learning rate, momentum and number of hidden neurons the stop training region can be easily identified as shown by Figure 2.7.

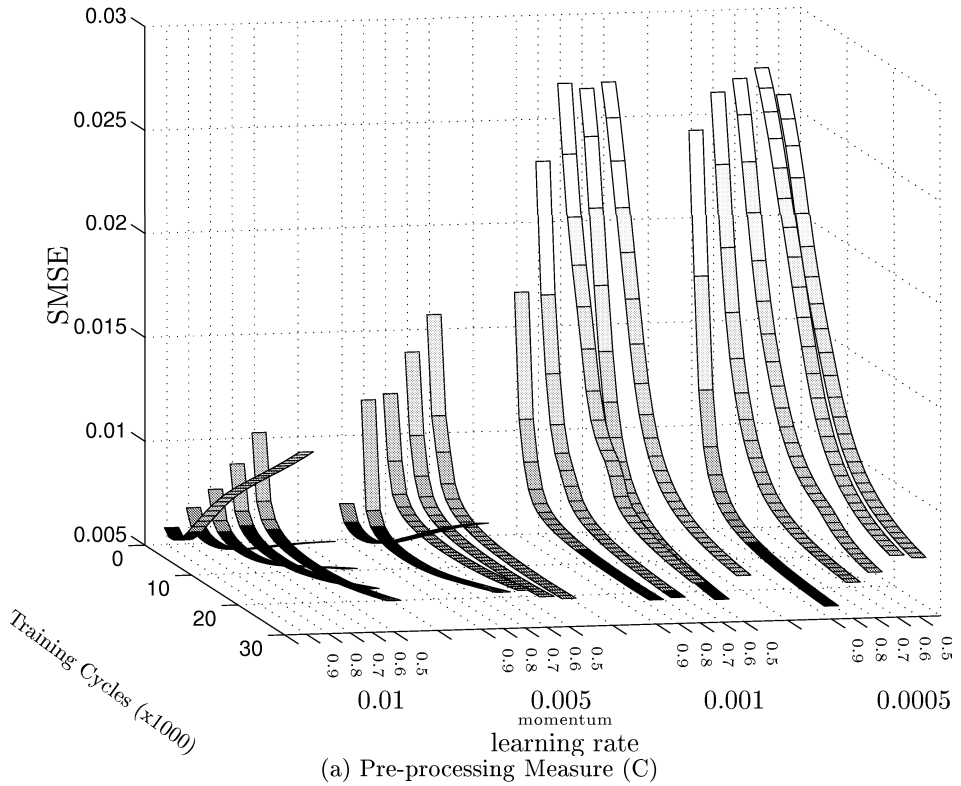
5.4.1 The 16 Transducer Data

The optimization of the neural networks using the 16 transducer sensor array data utilized databases tomo-1 and tomo-2, that is the 20mm artificial defect in composite GFRC(a). The results given here are summarized by four data sets formed from combinations of the two receiver arrangements, A and B, with two pre-processing techniques, one from the time domain, measure (C) and the other from the frequency domain, measure (D). The network topology for arrangement A was 48 input neurons, 30 hidden and 16 output neurons, while for arrangement B the number of input neurons was 36. The training data used defect off-set positions 1 and 2 with positions 3 and 4 used for the testing data. The standard BP learning algorithm was used with periodic update of weights.

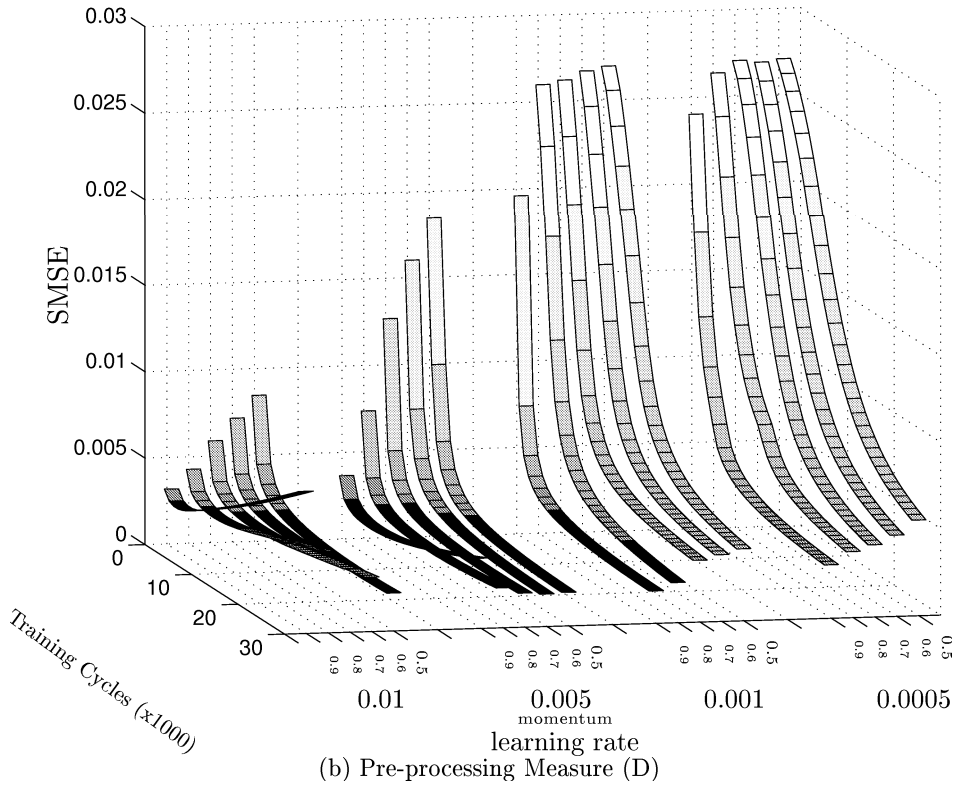
With learning rate values ranging from 0.0005 to 0.01 and momentum values from 0.5 to 0.9, twenty different neural networks were produced, each trained for 30,000 cycles (or 882 epochs) with testing errors collected at training intervals of 1,000 cycles (29 epochs). Figure 5.1 shows the SMSE of the testing data for the two pre-processing measures for transducer arrangement A; note that very similar plots were generated for arrangement B. Using the decision criterion previously mentioned, the optimum learning rate and momentum values selected are 0.001 and 0.9 respectively. These values will be used for all neural networks using data from the 16 transducer sensor array.

Next, for each combination of pre-processing measure and receiver arrangement twenty different neural network architectures with varying numbers of hidden neurons were used to train a neural network. The number of hidden neurons used ranged from 10 to 50. Note that the application of equation 2.45 indicates an appropriate number of hidden neurons to be between 4 and 32.

Despite there being a large change in the number of hidden neurons over the twenty



(a) Pre-processing Measure (C)



(b) Pre-processing Measure (D)

Figure 5.1: Learning Rate and Momentum Optimization for Arrangement A.

Table 5.2: Optimizing the number of hidden neurons.

Transducer Configuration A Pre-processing Measure (C)			Transducer Configuration B Pre-processing Measure (D)		
Hidden	Sensitivity	% Correct	Hidden	Sensitivity	% Correct
10	0.71	97.9	10	0.82	98.9
12	0.75	98.2	12	0.96	99.7
14	0.73	98.2	14	0.98	99.8
16	0.71	98.0	16	0.98	99.9
18	0.73	98.2	18	1.00	100
20	0.75	98.2	20	0.98	99.9
22	0.77	98.3	22	1.00	100
24	0.74	98.1	24	0.98	99.8
26	0.77	98.4	26	1.00	100
28	0.74	98.2	28	1.00	100
30	0.77	98.4	30	1.00	100
32	0.79	98.5	32	1.00	100
34	0.77	98.3	34	1.00	100
36	0.75	98.3	36	1.00	100
38	0.74	98.2	38	1.00	100
40	0.76	98.3	40	0.98	99.9
42	0.77	98.4	42	1.00	100
46	0.75	98.2	46	1.00	100
48	0.74	98.2	48	1.00	100
50	0.76	98.2	50	1.00	100

different architectures and thus number of trainable weights, all the networks were at an optimally trained state by 147 epochs (5,000 cycles). Table 5.2 shows the sensitivity and total percentage correct for the testing data for two of the four receiver arrangement and pre-processing measure combinations[1]. While for pre-processing measure (C) the optimum number of hidden neurons appears to be 32, the choice for measure (D) is less obvious with a whole range of values having exactly the same performance. The number of hidden neurons selected for both measures was 32 and will be used for all other experiments using any of the pre-processing measures unless otherwise stated.

The region to stop training was identified to be between 5,000 to 10,000 cycles, that is, 147 to 294 epochs. The stop training criteria will use 147 epochs for the 16 transducer data unless otherwise specified.

5.4.2 The 40 Transducer Data

Initial optimization of the 40 transducer data involved the tomo-7 database with targets formed to represent which one of the original four by four imaginary grid locations the 5mm defect is within. The learning rates used varied from 0.001 to 0.01, the momentum from 0.8 to 0.9, the number of hidden neurons from 10 to 100 (applying equation 2.45 gives a range from 6 to 61). The optimum values found were, learning rate 0.01, momentum 0.9 with 50 hidden neurons. The stop training region was upper bound by 200 epochs.

[1] Although not shown the neural network performance when using fewer than 10 hidden neurons was significantly reduced.

5.5 Initial Studies using the 16 Transducer Sensor Array

The main objective of the initial studies is to determine optimal specifications which will be used with the 40 transducer sensor array. This involves determining the optimum pre-processing technique with respect to neural network performance and the development of a neural network system to identify both defect location and size.

Before comparing the performance of neural networks trained with data using the various pre-processing techniques, a method of output image encoding and selection of the optimum data examples to be used for training data is required. Following the comparison of the pre-processing measures, investigation of other practical aspects, including the performance of other learning algorithms and application of the neural networks to validation data are performed. Finally an initial investigation of defect sizing by a neural network is given which leads to a methodology for the development of a neural network system for defect location and sizing.

The optimization of the neural networks using the 16 transducer sensor array data has determined the values for the training parameters and the number of hidden neurons. All neural networks given in this section will use the following values, learning rate 0.001, momentum 0.9 and the number of hidden neurons to be 32. The number of input neurons required depends on the transducer arrangement and the type of pre-processing measurement being used. Measurements (A) to (D) generate one measurement per waveform and so for arrangement A the number of inputs required is 48 while for arrangement B the number of inputs is only 36. However, measure (E3) for example, generates three measurements per waveform and so requires 144 inputs for arrangement A and 108 inputs for arrangement B. The number of outputs used in this section is set to 16, one for each of the imaginary grid locations, and is discussed in detail in Section 5.5.1. Except for those given in Section 5.5.4, all neural networks in this section use the standard BP algorithm with periodic update of weights.

5.5.1 Specification and Encoding of the Output Image

The most desirable situation is to have the neural network directly outputting an image of the scanning area. The simplest and most convenient solution is for each pixel within the reconstructed image to be represented by an output neuron.

But how many pixels should the reconstructed image have? This question can be answered in two ways. Firstly, it is necessary to determine the limitations of the complete system upon the output resolution and a detailed discussion about this aspect is given in Section 5.6. The second answer is that the number of pixels in the image and thus the number of output neurons are somewhat constrained by the number of defect locations used to collect the training data. For all the data collected using the 16 transducer sensor array a four by four imaginary grid was used to define the defect locations within the scanning area. Hence the reconstructed output image is a four by four pixel image requiring sixteen output neurons whereby each pixel represents a unique 20mm by 20mm portion of the scanning area.

The next step is to decide on the way a 'defect' and 'no defect' is represented at each pixel or output neuron. With reference to Section 2.7.3 the range for target values is limited and depends on the sigmoid function used. Both of the functions given in Figure 2.5 have been used with both the minimum and maximum values allowed used to represent a 'defect' target classification. And as expected the performance between these four alternatives was equivalent.

However the favoured method of encoding, used for the 16 transducer data, is to have a defect represented by 0 and a no defect by 1 which is then scaled for presentation to a neural network to give a 'defect' as -1 and a 'no defect' as 1 and of course uses the bi-polar sigmoid function. Output will be scaled back to the 0 to 1 range before calculating diagnostic performance.

5.5.2 Selection of the Training Data

The data collected with the 16 transducer sensor array, which was intended for training and testing data (tomo-1 and tomo-2), consists of four exemplars of a defect at each location. For each exemplar the defect had a slight off-set position within the grid

location as illustrated in Figure 4.1(b). As there are four exemplars per defect location, the obvious way to split them up into training and testing data sets is by having two exemplars each. Note that for the neural network optimization of the training parameters, defect positions 1 and 2 were used for training data and positions 3 and 4 for testing data, however, this may not be the optimum combination.

Again using databases tomo-1 and tomo-2 (the 20mm artificial defect with pre-processing measure (C) for receiver arrangement A and measure (D) for receiver arrangement B). Table 5.3 illustrates the performance achieved with the six unique combinations of defect positions used as training data. For arrangement A the variation in performance is most visible with the best results coming from the combination using defect positions 1 and 4 for training and 2 and 3 for testing. Arrangement B shows little improvement with the different defect positions.

Hence the optimum combination of defect exemplars, split equally between the training and testing data sets, is off-set positions 1 and 4 for the training data with positions 2 and 3 being used as testing data. Physically, positions 1 and 4 relate to the corners of each grid location, and effectively represent the points at which the grid locations change. This strengthens the author's belief that a BP neural network learns efficiently with training data whose examples are close to the class boundaries rather than examples which define the class centres.

For validation, improved class boundaries and thus generalization may be achieved by using three of the off-set positions for training data. Table 5.4 illustrates that the testing performance is improved for arrangement A. However, unless otherwise specified, all subsequent experiments will have training data consisting of defect positions 1 and 4 with testing data having defect positions 2 and 3.

5.5.3 Comparing the Pre-processing Techniques

Every pre-processing technique detailed in chapter 4 was applied to both databases tomo-1 and tomo-2. Thus, a neural network was trained and tested with data generated from one of the pre-processing methods for both arrangements.

Although the learning rate, momentum, number of hidden and output neurons

remained constant for all the neural networks, the number of input neurons varied according to the receiver arrangement and the pre-processing measurement used. Pre-processing measures (A) to (D) all give one value per waveform and so for arrangement A the number of inputs the NN has is 48 and for arrangement B the number of inputs is 36. Some of the pre-processing measurements produce more than one value per waveform, specifically pre-processing measures (E). With (E2) giving two values per waveform up to (E6) producing six values per waveform. Thus for neural networks using data sets with measures (E) the number of inputs for arrangement A will range from 96 to 288 and for arrangement B the range is from 72 to 216.

The increase in the number of neurons and thus trainable weights means that training was stopped towards the end of the stop training region, that is, at 10,000 cycles or 294 epochs. Table 5.5 illustrates the testing results for neural networks trained with all the different pre-processing measures for arrangements A and B. As can be seen the time-of-flight pre-processing measures (A to C) have best performance with arrangement A, while the frequency pre-processing measures (D and E) have the most effective results with arrangement B.

Comparing the confusion matrices and confidence maps for arrangement A, measurement (C) and arrangement B, measurement (D), (Figure 5.2 and Figure 5.3), illustrates that the former has reduced confidences both around the sides of the scanning area and within the middle four locations. The latter, however, has half as many reduced confidence locations all of which are located along the sides of the scanning area. Comparison of the threshold bounds of uncertainty illustrate that arrangement B measurement (D) gives the largest uncertain region and thus the highest generalization performance. Thus pre-processing measures (C) and (D) give the optimum performance with arrangements A and B respectively, with arrangement B combined with measurement (D) giving the overall best performance.

Additionally, investigation of combining pre-processing measures was also conducted. Combinations of time and frequency domain techniques were of special interest. However as shown by Table 5.6 no improvements were achieved.

Table 5.3: Comparing the selection of the training data.

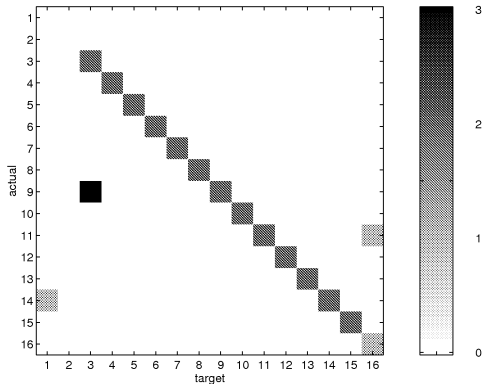
Off-set Positions		Arrangement A, Measure (C)		Arrangement B, Measure (D)	
Training	Testing	Sensitivity	% Correct	Sensitivity	% Correct
1 and 2	3 and 4	0.79	98.5	1.0	100
1 and 3	2 and 4	0.81	98.7	0.98	99.9
1 and 4	2 and 3	0.91	99.5	1.0	100
2 and 3	1 and 4	0.80	98.8	1.0	100
2 and 4	1 and 3	0.81	98.9	1.0	100
3 and 4	1 and 2	0.82	98.9	0.98	99.9

Table 5.4: Additional comparison of the training data.

Off-set Positions		Arrangement A, Measure (C)	
Training	Testing	Sensitivity	% Correct
2, 3, 4	1	0.85	99.1
1, 3, 4	2	0.96	99.7
1, 2, 4	3	0.96	99.7
1, 2, 3	4	0.85	98.9

Table 5.5: Comparing the pre-processing measurements.

Pre-processing Measure	Transducer Arrangement A		Transducer Arrangement B	
	Sensitivity	% Correct	Sensitivity	% Correct
A	0.61	97.6	0.61	97.6
B	0.60	97.4	0.58	97.4
C	0.87	99.2	0.77	98.5
D	0.90	99.4	0.99	99.9
E2	0.99	99.9	0.99	99.9
E3	1.0	100	1.00	100
E4	0.97	99.8	0.99	99.9
E5	0.98	99.9	1.00	100
E6	0.96	99.7	1.00	100

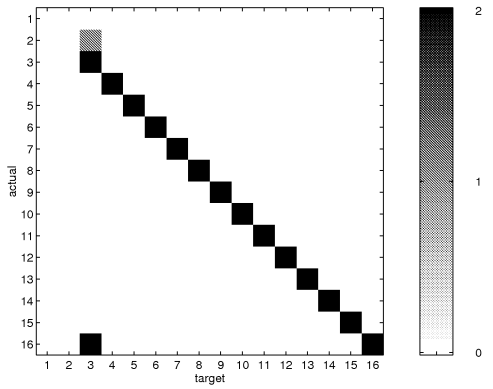


(a) Confusion matrix

0.00	1.00	0.00	1.00
0.00	1.00	1.00	0.67
1.00	1.00	0.67	1.00
1.00	1.00	1.00	0.50

(b) Confidence map

Figure 5.2: Classification for arrangement A with measurement(C).



(a) Confusion matrix

0.00	1.00	1.00	1.00
−0.50	1.00	1.00	1.00
1.00	1.00	1.00	1.00
1.00	1.00	1.00	0.00

(b) Confidence map

Figure 5.3: Classification for arrangement B with measurement(D).

Table 5.6: Comparison of combined pre-processing measurements.

Pre-processing Measure	Transducer Arrangement A		Transducer Arrangement B	
	Sensitivity	% Correct	Sensitivity	% Correct
C and D	0.92	99.5	0.96	99.8
A, C and D	0.86	99.2	0.93	99.6
A, B, C and D	0.82	98.9	0.91	99.5

5.5.4 Comparing Learning Algorithms

Although most of the neural networks detailed in this thesis use the standard BP learning algorithm, two other algorithms, both of which are variants of the BP algorithm, have been investigated. They are referred to as the Quickprop (QP) and Radial Basis Function (RBF) algorithms (details given in Section 2.3.1 and Section 2.5 respectively). The comparison between the learning algorithms could be judged on many different aspects, for example, computational time, speed of convergence, memory requirements, and generalization ability. However, the purpose of this investigation is not to give a complete comparison of the different learning algorithms but to decide if improved generalization performance can be achieved with another learning algorithm.

For consistency, as the PlaNet simulation does not implement QP or RBF algorithms, all three will be simulated with the NeuralWorks package. The comparison of the learning algorithms used the same training and testing data sets, arrangement B with pre-processing measure (D), trained for up to 15,000 cycles and tested every 1,000 cycles with the best testing network saved. The same learning rate, momentum and number of hidden neurons are used for all three algorithms.

Table 5.7 shows the five performance measures calculated from the diagnosis metrics with varying threshold values for the three different learning algorithms under testing for the best performing simulation out of the five performed per learning algorithm. The bounds of uncertainty for the BP network is 0.2 to 0.7, for the QP network is 0.4 to 0.6 and for the RBF network is 0.7. This suggests that the BP neural network has potentially the greatest generalization performance.

The performance of the QP and RBF neural networks are improved by using different values of learning rate and momentum and varying the number of hidden neurons used. For example, with 40 hidden neurons the bounds of uncertainty are 0.2 to 0.5 for QP and 0.7 to 0.8 for RBF. However, the performance achieved by the BP neural network was not improved upon.

Table 5.7: Comparison of the learning algorithms.

NN	Measure	Threshold						
		0.2	0.3	0.4	0.5	0.6	0.7	0.8
BP	Sens	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Spec	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	PosPV	1.00	1.00	1.00	1.00	1.00	1.00	0.94
	FalAR	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	% Corr	100.00	100.00	100.00	100.00	100.00	100.00	99.63
QP	Sens	0.97	0.97	1.00	1.00	1.00	1.00	1.00
	Spec	1.00	1.00	1.00	1.00	1.00	1.00	0.99
	PosPV	1.00	1.00	1.00	1.00	1.00	0.94	0.89
	FalAR	0.00	0.00	0.00	0.00	0.00	0.00	0.01
	% Corr	99.82	99.82	100.00	100.00	100.00	99.63	99.26
RBF	Sens	0.69	0.84	0.88	0.97	0.97	1.00	1.00
	Spec	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	PosPV	1.00	1.00	1.00	1.00	1.00	1.00	0.97
	FalAR	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	% Correct	98.16	99.08	99.26	99.82	99.82	100.00	99.82

5.5.5 Validating with Different Defect Sizes

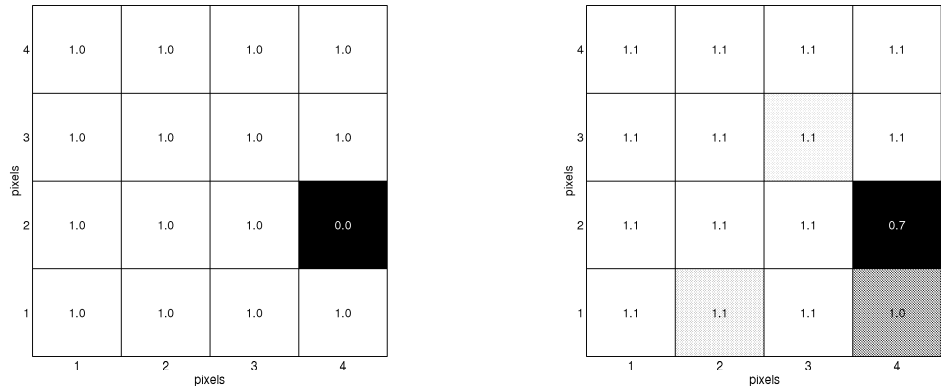
The previous sections have investigated various aspects of the neural network to produce an optimal neural network system from the available training and testing data. Given such a system an investigation can be conducted to determine how well the system can generalize, that is, classify completely 'unseen' or validation data. In this application the neural network has been trained and tested with artificial defects in the form of 20mm diameter circular holes.

The extent of validation may range from simply testing with data from various sizes of artificial defects (circular holes) contained within the same material as used for training, to testing with data from completely different types of defects contained within different materials. The validation performed here involves using a neural network trained and tested with 20mm diameter artificial defects in the GFRC(a) material and validated with examples of a 5mm diameter artificial defect, again in the GFRC(a) material. Further validation is given in Section 5.7.

Given a neural network, trained using standard BP with a learning rate of 0.001,

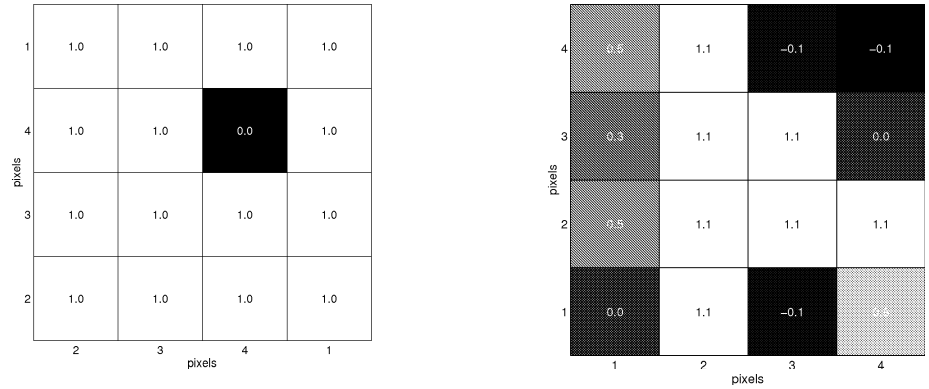
a momentum of 0.9, periodic update of weights and an architecture of 48 inputs, 32 hidden and 16 output neurons. The training data is taken from arrangement A, using pre-processing measure (D) with defect positions 1 and 4 (positions 2 and 3 are used for testing). This neural network is validated with data set tomo-3, a 5mm diameter circular hole artificial defect. Figure 5.4 to Figure 5.6 illustrate both the target location of the 5mm defect and the output image from the neural network[2]. The performance measures are 2 TPs, 457 TNs, 55 FPs and 30 FNs which indicates that there are only two examples were the 5mm defect has been correctly located. Detailed inspection of all the validation images shows that most of the neural network images have difficulty in even getting close to the actual defect position, shown in Figure 5.6. In general, mis-classified defects produce images with pixels close to the right hand side (where the four receivers are physically positioned) are active, as demonstrated by Figure 5.5 and Figure 5.6.

[2]Both the target and neural network images are displayed with the actual numerical values shown in the middle of each pixel. The colour map is relatively grey scaled for the given image with black indicating the lowest numerical value and white the highest. Note that the values of the output image extend beyond the 0 to 1 range because the neural network simulator internally scales the data to fall within the 0.1 to 0.9 range; thus in re-scaling validation data output, output responses may range from -0.1 to 1.1.



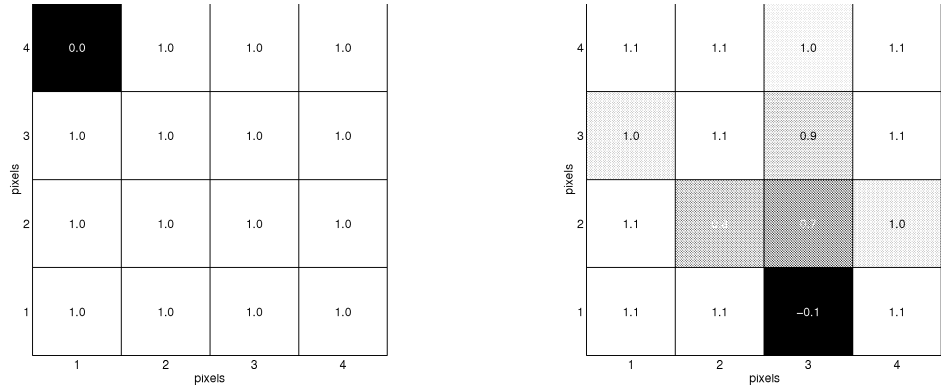
(a) Target (b) Image

Figure 5.4: Example of good classification of the 5mm defect's location.



(a) Target (b) Image

Figure 5.5: Example of a confused classification of a 5mm defect.



(a) Target (b) Image

Figure 5.6: Typical classification trained with 20mm defect, validated with 5mm defect.

5.5.6 Varying the Defect Size

All of the previous neural networks were trained and tested with data which used the 20mm diameter circular hole. This meant that the neural networks' task was purely the location of the defect within the scanning area, with no indication of the defect's actual size. However, it is most desirable that not only the location of defects be determined but also an estimate of the defect's actual size. The simplest method of achieving defect location and sizing by a neural network system is to separate the problem into two tasks and have one neural network for each, that is, one NN to determine defect size and another to determine defect location.

For the sizing neural network, artificial defects of size 20mm, 5mm and 1mm (data sets tomo-1, tomo-3 and tomo-4) were used as training and testing data and defects of size 10mm as validation data (data set tomo-5). Both the training and testing data contained one of the defect off-set positions per location giving sixteen examples of each defect size plus an example with no defect per defect size. Pre-processing measure (D) was used and the network architecture was 48 inputs with 32 hidden and 1 output neuron. Encoding the size as a single numerical output ranging from 0 for no defect to 1 for a 20mm defect with 0.2 representing a 1mm defect, 0.4 representing a 5mm defect and 0.6 indicating a 10mm defect. Results using the validation data of the 10mm defect size were very promising with the neural network classification lying in between the decision threshold boundaries of 0.5 and 0.7 for three of the four examples of the 10mm defect (specifically the output classifications were as follows 0.585408, 0.648379 and 0.542439), with the fourth example giving an output of 0.466438 which represents a 5mm defect.

For the location of defects of various sizes, Section 5.5.5 has demonstrated that a neural network trained with 20mm defect is unable to correctly identify the location of a 5mm defect. And so the approach of training a single neural network with several defect sizes is investigated. Again artificial defects of size 20mm, 5mm and 1mm were used for training and testing and again pre-processing measure (D) was used making the number of network inputs to be 48. As with all the other neural networks trained for defect location, sixteen output neurons were used to represent the sixteen imaginary

grid locations within the scanning area. Note that both periodic and continuous updating of weights were investigated, with continuous update giving a slight improvement. In addition, the number of hidden neurons was varied in the search for optimum performance, initially the number used was 32 and unfortunately improved performance was not achieved with a larger number. As indicated by Table 5.8 which gives the testing results, the neural network has difficulty in correctly locating the defects (low sensitivity and positive predictive value). Inspection of the confusion matrices, given in Figure 5.7, for the testing examples of the three different defect sizes, suggests that while the neural network is attempting to locate the smaller sized defects their inclusion in the training and testing data sets has reduced correct classification of the 20mm defects (and has brought about a significant increase in FP when classifying the 20mm defects).

Another neural network was trained but with only the 20mm and 5mm defects. The performance metrics for testing are given in Table 5.9. The confusion matrices are shown in Figure 5.8 and illustrate an improvement in classification in terms of false positives but comparatively fewer true positive classifications for both the 20mm and 5mm defects. The results from the previous two neural networks indicate a difficulty in the location of defects of different size by the use of training data consisting of different defect sizes.

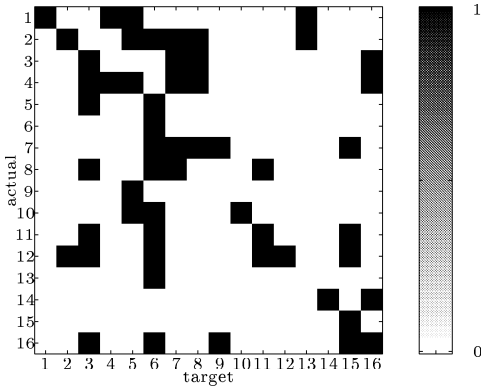
A system which separates the task of defect location and sizing may suffer problems when presented with situations which have several defects present within the scanning area. The most obvious being, which defect is the sizing neural network referring to. A possible solution is to increase the output resolution of the location neural network allowing it to indicate both location and size. Unfortunately, such a system might suffer from similar problems to that previously demonstrated with the neural networks trained to locate defects of various size.

Thus the methodology for defect location and sizing, believed by the author to be the optimal solution given the previous results, is to train a neural network to locate a small defect in a large number of positions within the scanning area. By assuming that the changes in the ultrasonic raypaths brought about by a large defect will be

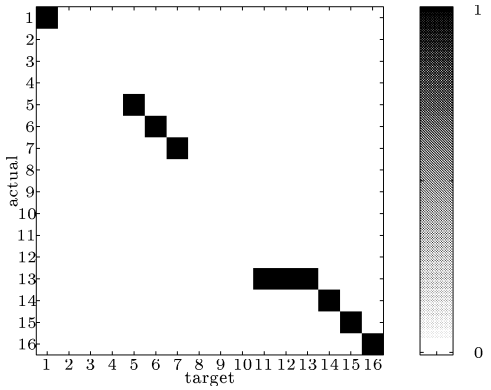
similar to the collective changes of several small defects positioned adjacently at the same location as the larger defect, then a large defect may be considered to be simply a collection of smaller defects. By this assumption we can infer that the trained neural network should be able to detect larger defects than it was trained with, by indicating a number of smaller defects next to one another. This methodology will be used in the data collection for the 40 transducer sensor array, detailed in the next section.

Table 5.8: Location of defects of size 20mm, 5mm and 1mm.

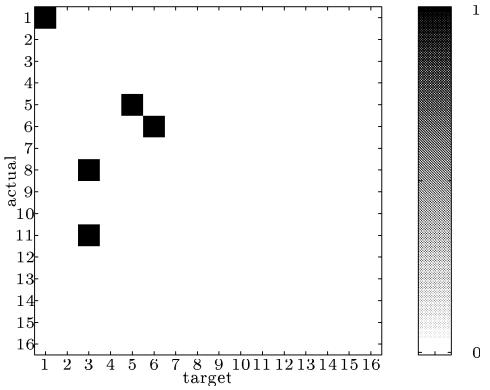
	Threshold						
Measure	0.2	0.3	0.4	0.5	0.6	0.7	0.8
TP	11	15	21	23	31	34	38
TN	744	735	729	715	694	681	654
FP	24	33	39	53	74	87	114
FN	37	33	27	25	17	14	10
Sens	0.23	0.31	0.44	0.48	0.65	0.71	0.79
Spec	0.97	0.96	0.95	0.93	0.90	0.89	0.85
PosPV	0.31	0.31	0.35	0.30	0.30	0.28	0.25
FalAR	0.03	0.04	0.05	0.07	0.10	0.11	0.15
% Corr	92.52	91.91	91.91	90.44	88.85	87.62	84.80



(a) 20mm defects



(b) 5mm defects

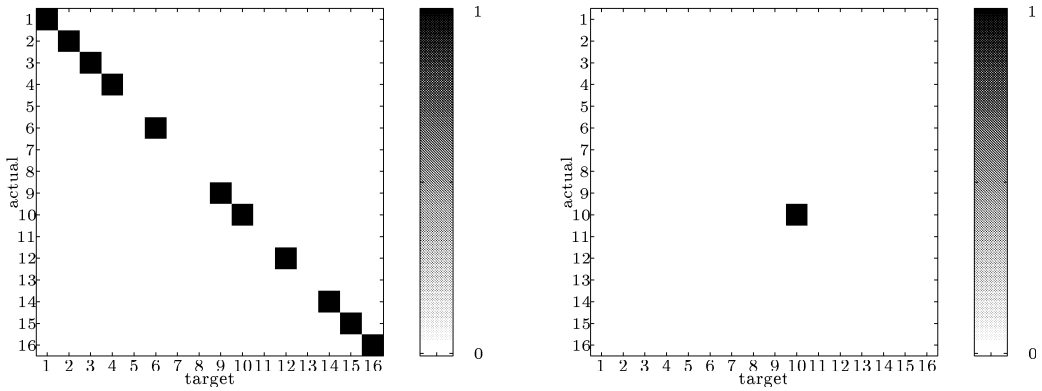


(c) 1mm defects

Figure 5.7: Confusion matrix for the three defect sizes.

Table 5.9: Location of defects of size 20mm and 5mm.

	Threshold						
Measure	0.2	0.3	0.4	0.5	0.6	0.7	0.8
TP	7	8	10	12	14	15	17
TN	512	512	512	512	511	511	506
FP	0	0	0	0	1	1	6
FN	25	24	22	20	18	17	15
Sens	0.22	0.25	0.31	0.38	0.44	0.47	0.53
Spec	1.00	1.00	1.00	1.00	1.00	1.00	0.99
PosPV	1.00	1.00	1.00	1.00	0.93	0.94	0.74
FalAR	0.00	0.00	0.00	0.00	0.00	0.00	0.01
% Corr	95.40	95.59	95.96	96.32	96.51	96.69	96.14



(a) 20mm defects (b) 5mm defects

Figure 5.8: Confusion matrix for the two defect sizes.

5.6 The 40 Transducer Sensor Array

The specific receiver arrangement for the 40 transducer sensor array, used for data collection, is illustrated in Figure 4.6(b). Given the tomographic data sets collected from the 40 transducer sensor array, detailed in Table 4.4, initial neural networks are trained to produce 4 by 4 pixel images. After examination of the fundamental limitations upon output resolution, several methods for maximizing the output resolution abilities of the system are detailed.

5.6.1 Initial Imaging

The database, tomo-7, consists of 256 unique locations of the 5mm artificial defect within the scanning area. Thus the output resolution can be as high as a 16 by 16 pixel image where one pixel represents a 5mm by 5mm area of the scanning area. To generate such a resolution the neural network would require 256 output neurons, and as detailed in Section 5.6.2, a single neural network with this many output neurons proves difficult to train.

To allow the optimization of neural network training parameters and the selection of the optimal training data set, the 256 defect locations have been encoded onto 32 outputs. The first sixteen outputs specify which one of the original sixteen imaginary grids the 5mm defect is positioned while the last sixteen outputs indicate in a similar fashion the location within the selected grid (see Figure 4.2).

However, the successful training of a neural network which used just the 32 encoded outputs also proved difficult, with respect to achieving desirable performance on testing. Hence, for the optimization of the neural network training parameters and for the selection of optimal training data, just the first sixteen encoded outputs were used. With only the first sixteen outputs, specifying the original 20mm by 20mm grid location the 5mm defect is within, neural networks were successfully trained allowing both the optimization of training parameters but also the investigation of optimal training data, as detailed below.

Given the optimized neural network, as described in Section 5.4.2, selection of the training data is performed. The database tomo-7 contains four exemplars of a defect at

each location. Thus the database is divided into two, with both the training and testing data sets having two different exemplars of each location. Assuming the database is initially divided into four groups, with each group consisting of one exemplar of a defect at every location and one no defect exemplar. These four groups are then labelled numerically, 1 to 4. The combinations of these groups of exemplars, to form training and test data sets, are illustrated in Table 5.10 along with the testing performance of a trained neural network. Although the performances are all very high, the best performance is with a combination of groups 1 and 3 for training data and 2 and 4 for testing (highest sensitivity and percentage correct). This combination for training and testing data sets will be used for all subsequent neural networks.

Table 5.10: Comparing the selection of the training data for the 40 transducer sensor array.

Data Sets		Performance Measures				
Training	Testing	Sens	Spec	PosPV	FalAR	% Corr
1 and 2	3 and 4	0.92	0.99	0.92	0.00	99.0
1 and 3	2 and 4	0.93	0.99	0.94	0.00	99.2
1 and 4	2 and 3	0.92	0.99	0.94	0.00	99.1
2 and 3	1 and 4	0.83	0.98	0.79	0.01	97.6
2 and 4	1 and 3	0.79	0.98	0.80	0.01	97.5
3 and 4	1 and 2	0.73	0.98	0.71	0.02	96.4

5.6.2 Increasing the Output Resolution

All of the previous neural networks have produced images of resolution 4 by 4 pixels. The initial reason for producing a 4 by 4 pixel image, for the 16 transducer sensor array, was because it directly reflected the different number of defect locations. As previously discussed, the output resolution is effectively determined by the number of defect locations within the training data set and the limitations of the tomographic system. Thus, as the number of unique defect locations, for data set tomo-7, has increased to 256 the output resolution may potentially be increased to a resolution of 16 by 16 pixels. However, before increasing the resolution, an examination of all the limitations upon the system with respect to resolution ability is performed.

5.6.2.1 Limits Upon Output Resolution

There are two main limiting factors upon the output resolution, the first is caused by the practical limits of the sensor array while the second is due to a combination of the nature of neural network learning and the training data.

Limitations of Conventional Tomography

There are two main conventional methods of tomographic reconstruction for ultrasound. The first, transformation methods, are based in the frequency domain and use Fourier transforms, while the second, series-expansion methods, are based in the spatial domain and use iterative algorithms [50]. Both methods have several factors which affect the maximum possible output resolution; however, the relationship between the transducer arrangement and the output resolution is of most interest.

Transformation methods usually require a symmetric arrangement of transducers, giving an even distribution of raypaths throughout the scanning area. Typically, a number of projections are sampled at different angles through the scan area, each projection consisting of a series of parallel raypaths [108]. The distance between two adjacent parallel raypaths limits the frequency bandwidth of the Fourier transform convolutions, determined by the Nyquist sampling theorem. Thus, to avoid aliasing, the spatial frequency, which gives the output resolution, is bound by the raypath separation,

which is determined by the transducer arrangement.

Series-expansion methods may have an asymmetric arrangement of transducers, giving a possibly uneven distribution of raypaths throughout the scan area (i.e. an uneven raypath density). The scan area undergoes discretization, forming an array of pixels, the resolution of which is determined by the total number of raypaths [109]. An iterative algorithm takes each raypath in turn, and minimizes the values of the pixels it traverses using least squared error between the experimental data and the projected image. Under-determined pixels, traversed by few or no raypaths, are smoothed by taking the average of surrounding pixels [110]. Because an asymmetric arrangement is allowed, the smallest detectable defect size may vary over the scan area. Hence, intuitively, reducing the size of areas within the scan area, which have few or no raypaths, will maximize the achievable output resolution over the entire scan area.

In addition to the relationship between the output resolution and the transducer arrangement, there are other factors which also affect the maximum possible output resolution for both conventional methods. These include the beam width of the ultrasound raypaths and the maximum frequency component within the ultrasonic transients.

It is the author's view that a neural network method of tomographic reconstruction with its asymmetric arrangement of transducers and learning in the spatial domain (target classifications relate directly to the output image) will have similar resolution constraints to that of the series-expansion methods; specifically, the output resolution will be limited by the following three factors:

- Small areas of the scan area which contain no or few raypaths (no raypath coverage); these will vary according to the transducer arrangement but are difficult to visualise experimentally.

- The beam width of the ultrasound raypaths; this is determined by the contact area of the couplant used between the pinducers and the sample material.

- The maximum frequency component within the ultrasonic transients, this governs the smallest ultrasonic wavelength, which in turn determines the minimum resolution via the Nyquist sampling theorem (at half the wavelength). The fre-

quencies relate to the physical characteristics of the pinducers and will remain constant irrespective of the transducer arrangement.

Hence, to increase the output resolution for the neural network method of tomographic reconstruction, the number of raypaths must be increased to reduce the possibility of small areas within the scan area having no raypath coverage. The need to increase the number of transducers in the sensor array gave rise to the instrumentation shown in Figure 3.10, which allows configurations with up to 40 transducers, eight of which may act as receivers. With the experimental apparatus detailed in Section 3.3.2.2, the detectable defect size will be lower bound by the ultrasonic beam width, defined by the rubber coupling device which is 3.5–4mm in diameter, and half the wavelength of the highest frequency component of the ultrasonic transients, which gives 2mm–5mm with the highest frequency, shown by Figure 4.14, at approx. 0.5MHz (Note that typical wavelengths are 4mm–10mm for a longitudinal wave travelling through composite material, with velocities ranging from 2000ms^{-1} to 5000ms^{-1}). Hence a 5mm diameter defect should be detectable.

Limitations of the Training Data and the Neural Networks

Neural networks limit the possible output resolution in two ways. Firstly the number of outputs possible and thus the output resolution is directly related to the quantity and variation of the collected data. Even with the semi-automated data collection method previously detailed, the time constraints limit the amount of available data and thus the maximum resolution.

The second limitation is caused by the imbalance found in each training example with an output image as the target, typically the target consists of a small localized defect within a much larger scan area. Thus the targets for any given example are mainly 'no defect'. Conventional methods of balancing class representations within a training data set are ineffective as for each example there are many more targets of 'no defect' than there are for a 'defect'. This is the most significant neural network limitation and is referred to as the 'image biasing' problem.

5.6.2.2 Reducing Image Biasing

Unfortunately a problem occurs with the training of a neural network which has a large number of output neurons, of which only one is active for a given input pattern. With database tomo-7 there are 256 defect locations and thus 256 output neurons. For each input pattern, 1 output is active, indicating the location of the defect, with the other 255 outputs inactive, indicating the lack of presence of a defect. The neural network will tend to concentrate its learning on the 255 neurons lack of presence of a defect rather than the 1 neuron showing a defect. This effectively results in a trained neural network which has learnt where there is no defect for each pattern but does not attempt to indicate where the defect is. This effect is termed 'image biasing' and a number of methods and strategies have been devised to reduce the image biasing problem.

Modified Learning Rate

The first method devised to reduce the effects of image biasing consists of dynamically altering the learning rate during training. Effectively each output neuron will have its own learning rate which is altered for each input pattern according to the target output. For instance, if a target output represents a 'defect' then the learning rate for that output neuron is set to the normal value, however, if a target output represents a 'no defect' then the learning rate for that output neuron is significantly reduced. This is performed for every output neuron for every input pattern presented to the neural network during training and assumes per pattern update of weights.

Given equation 2.17 the modification to the learning rate is shown by

$$\Delta w_{kj}^{t+1} = \eta_{\text{mod}} \left(t_k - o_k\right) o_k \left(1 - o_k\right) y_j \tag{5.2}$$

where η_{mod} is the modified learning rate and is determined by

$$\begin{aligned} \eta_{\text{mod}} &= \eta \iff t_k \Rightarrow \text{defect} \\ \eta_{\text{mod}} &= \frac{\eta}{g} \iff t_k \Rightarrow \text{no defect} \end{aligned} \tag{5.3}$$

with g defining the fractional reduction of the learning rate for output neurons representing a no defect target.

As shown in Table 5.11, a number of values of g have been investigated and although the number of true positives is approaching the maximum value, the number of false positives is by comparison too large for a reliable system. This is reflected by the low positive predictive values achieved.

Divide and Conquer

The second method devised to reduce the problem of image biasing is referred to as the divide and conquer approach. Here the complete image area is divided into smaller sub-images with each sub-image being generated by a separate neural network. Thus the number of outputs per neural network is significantly reduced without affecting the overall resolution of the complete image.

The complete image is divided into sixteen sub-images. Each sub-image directly relates to the portion of the scan area defined by the imaginary four by four grid. With each sub-image consisting of a 4 by 4 pixel image whereby a pixel represents a 5mm by 5mm area of the complete scan area. This gives a system which requires sixteen neural networks, each one generating a 4 by 4 pixel sub-image, which, when combined with the others, produces a 16 by 16 pixel image of the scan area.

For the divide and conquer technique, there were a number of ways to generate the individual training data sets for each sub-image neural network. Initially the sixteen individual training data sets were constructed by filtering out (removing from the data set) examples of defects which were outside the sub-image area. Thus the sixteen neural networks have been trained and tested with only the defects within its sub-image. The testing results of the sixteen neural networks is given in Table 5.12. As shown the performance indicates both a good detection of defects and overall percentage correct classification. However, when the testing examples of all 256 defect locations were presented to all sixteen neural networks and the images combined, the result was a rather poor quality image. The author believes that the reason for this result was because each neural network needs to learn both when a defect was within its sub-

Table 5.11: Modified learning rate.

Perform. Measures	Value of $1/g$						
	0.01	0.007	0.005	0.003	0.001	0.0009	0.0007
TP	452	470	479	480	483	475	491
TN	123703	118261	120061	113807	91941	90793	82736
FP	7369	12811	11011	17265	39131	40279	48336
FN	60	42	33	32	29	37	21
Sens	0.88	0.92	0.94	0.94	0.94	0.93	0.96
Spec	0.94	0.90	0.92	0.87	0.70	0.69	0.63
PosPV	0.06	0.04	0.04	0.03	0.01	0.01	0.01
FalAR	0.06	0.10	0.08	0.13	0.30	0.31	0.37
% Corr	94.35	90.23	91.61	86.85	70.24	69.36	63.25

Table 5.12: The testing of the sixteen networks for divide and conquer, trained with data sets containing only examples of defects within the sub-image.

Network	Sensitivity	% Correct	Network	Sensitivity	% Correct
1	0.90	99.5	9	1.00	100
2	0.97	99.6	10	0.97	99.8
3	1.00	99.8	11	1.00	100
4	1.00	100	12	0.91	99.5
5	1.00	100	13	1.00	100
6	1.00	100	14	1.00	98.7
7	0.96	99.8	15	0.97	99.8
8	0.93	99.5	16	1.00	99.5

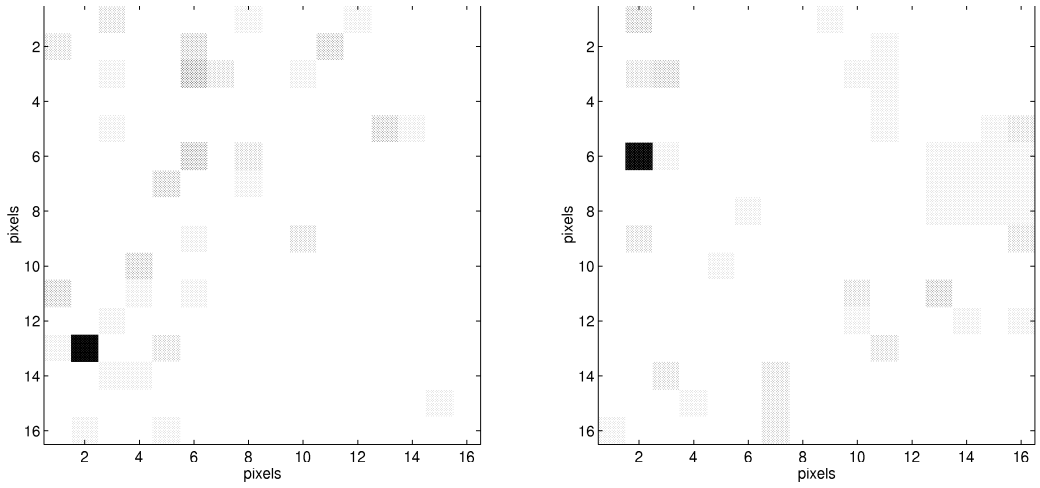
Table 5.13: The testing of the sixteen networks for divide and conquer, trained with the new data sets containing examples of both repeated defects within the sub-image area and non-repeated defects outside the sub-image area.

Network	Sensitivity	% Correct	Network	Sensitivity	% Correct
1	0.88	99.3	9	0.97	99.8
2	0.94	99.6	10	0.94	99.6
3	0.90	99.5	11	0.88	99.3
4	1.00	100	12	0.84	98.9
5	1.00	100	13	1.00	100
6	0.97	99.8	14	0.88	99.3
7	0.84	99.1	15	0.97	99.8
8	0.84	99.1	16	1.00	100

image and also when the defect was outside its sub-image. This has been achieved by including all the defect examples, but examples which were outside the given neural network's sub-image area had the target outputs altered to indicate a no defect sub-image. Doing this created a data set which was unbalanced and so repetition of the examples which have defects within the sub-image was required to generate a balanced data set.

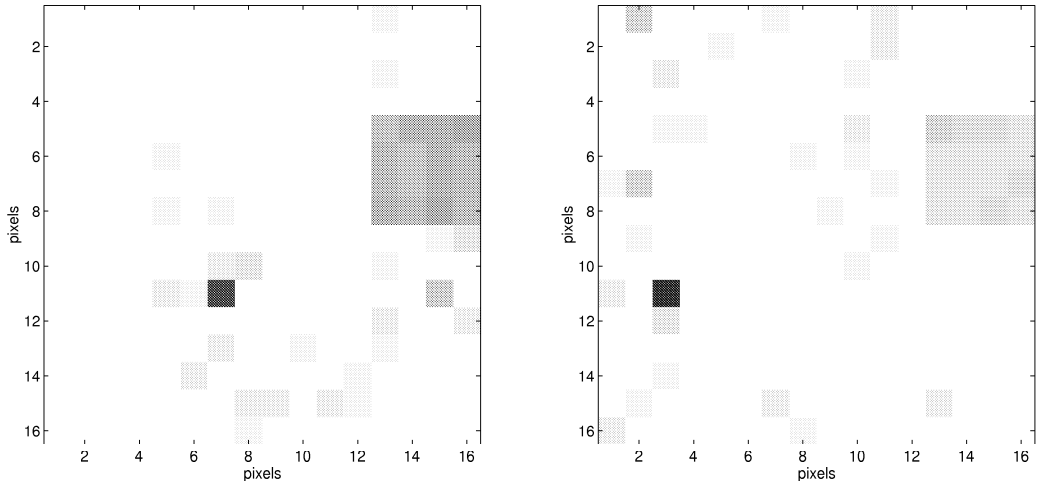
The sixteen neural networks trained with these new data sets still give good results when tested with defects located within the sub-image area, as indicated by Table 5.13, and are better able to generate good quality complete images, as shown in Figure 5.9. Note that the two images shown are generated by the sixteen neural networks (no target images) and illustrate correct location of two differently positioned 5mm defects; in both images the defect is indicated by the black pixel.

An additional problem with using several neural networks to generate one image, is to ensure that each neural network is trained to the 'same' state whereby the output values generated for a 'defect' and 'no defect' classification are the same for all sixteen networks. This problem is best illustrated in Figure 5.10, where the complete image is scaled between 0 and 1, and the neural network at grid location 14 appears to produce higher values for a 'no defect' classification than the others.



(a) (b)

Figure 5.9: Combined images for divide and conquer testing.



(a) (b)

Figure 5.10: Combined images for divide and conquer testing with one network producing different values for 'defect' and 'no defect' classifications.

5.7 Neural Network Images

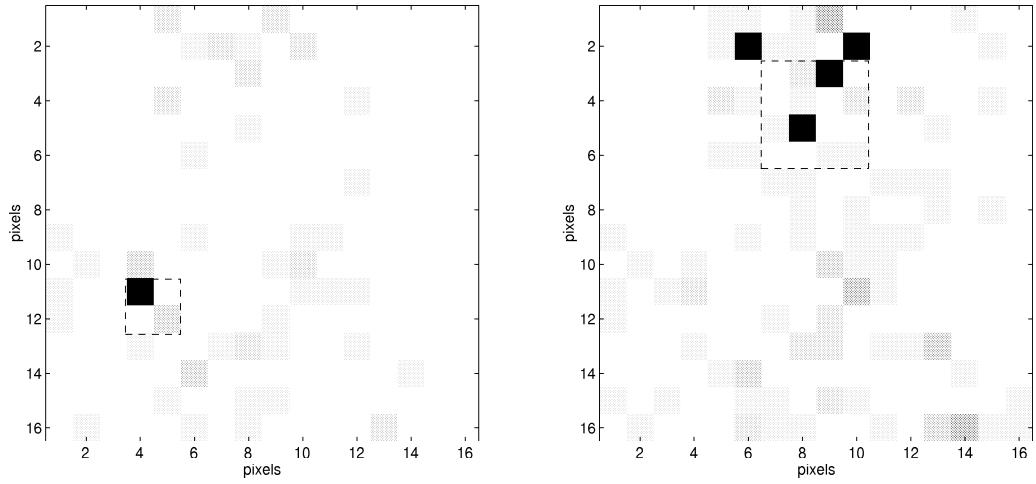
Initial validation of the 16 transducer sensor array neural networks has already been conducted, that is, training and testing with a 20mm diameter defect and validating with a 5mm defect. The validation procedure is now extended to cover neural networks which have been trained with either the 16 or 40 transducer sensor array data, and will include validation data taken from different composite materials containing various defect types. Five different validation cases have been examined and are detailed below in increasing order of variation between the material used to train the neural networks and the material used to collect the validation data.

5.7.1 Artificial Defects in Glass Fibre Composite

The neural network trained with the 16 transducer sensor array data has been previously (see Section 5.5.5) unsuccessfully validated with various different defect sizes within the glass fibre composite material GFRC(a). However, the sixteen divide and conquer neural networks, which generate the 16 by 16 pixel images, have been sucessfully tested with the 5mm artificial defects, again within the glass fibre composite. Thus, the sixteen neural networks are now to be validated with artificial defects of different size to that used to train the networks. Specifically, data set tomo-8 was used, which contains examples of a 20mm and a 10mm circular hole defect within the GFRC(a) material. Figure 5.11 illustrates the images for a 10mm and 20mm defect with the actual defect position and size indicated by a dashed square outline. Although the neural networks were only trained with a 5mm defect, the images do suggest both the location and the size of the larger defects. Figure 5.12 and Figure 5.13 illustrate two additional 10mm and 20mm defect images.

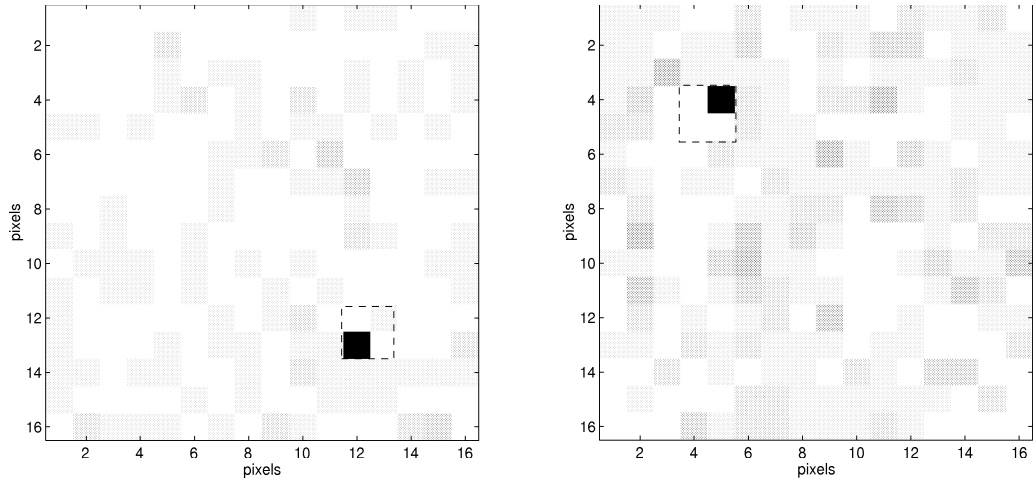
5.7.2 Two Artificial Defects in Glass Fibre

The sixteen neural networks have been validated with a glass fibre reinforced composite GFRC(a) sample containing two 5mm circular hole defects. Figure 5.14(a) illustrates their relative position within the scanning area while Figure 5.14(b) shows the reconstructed image. Although the exact location of the two defects has not been indentified it is clear that two defects are present.



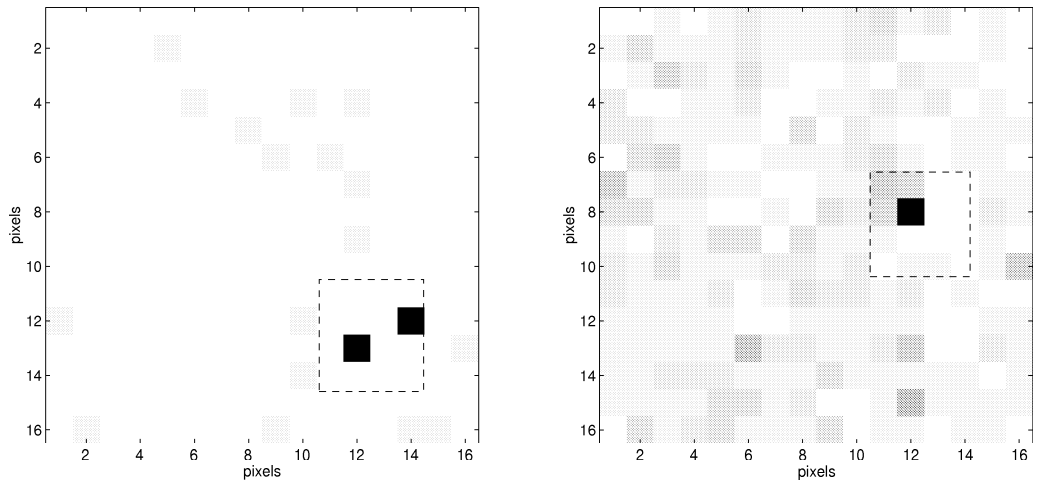
(a) 10mm defect (b) 20mm defect

Figure 5.11: Images of a 10mm and 20mm artificial defect.



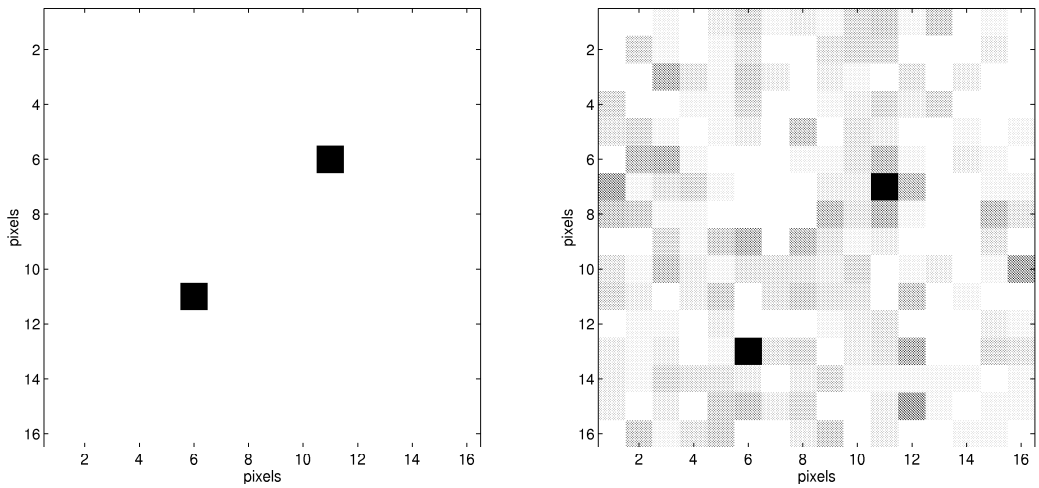
(a) 10mm defect (b) 10mm defect

Figure 5.12: Other example images of the 10mm artificial defect.



(a) 20mm defect (b) 20mm defect

Figure 5.13: Other example images of the 20mm artificial defect.

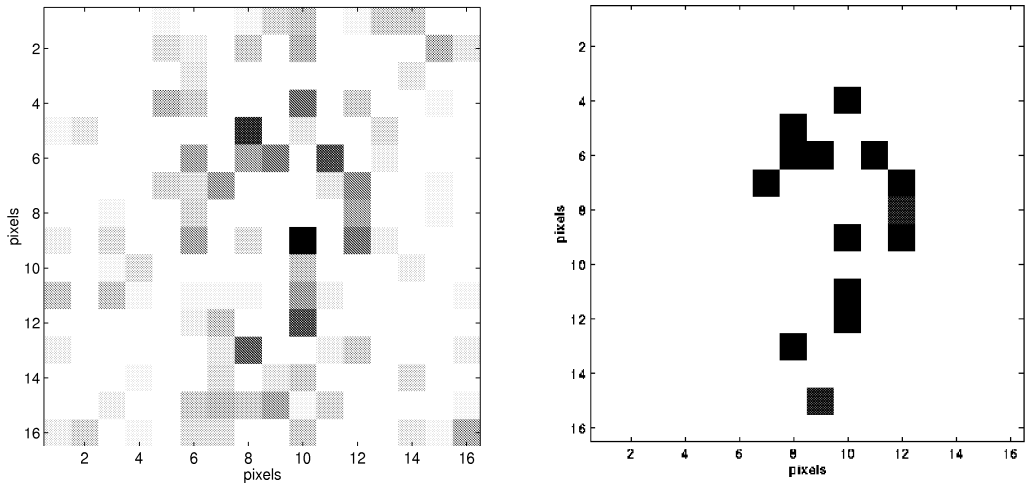


(a) Target (b) Image

Figure 5.14: Image of a sample containing two 5mm diameter defects.

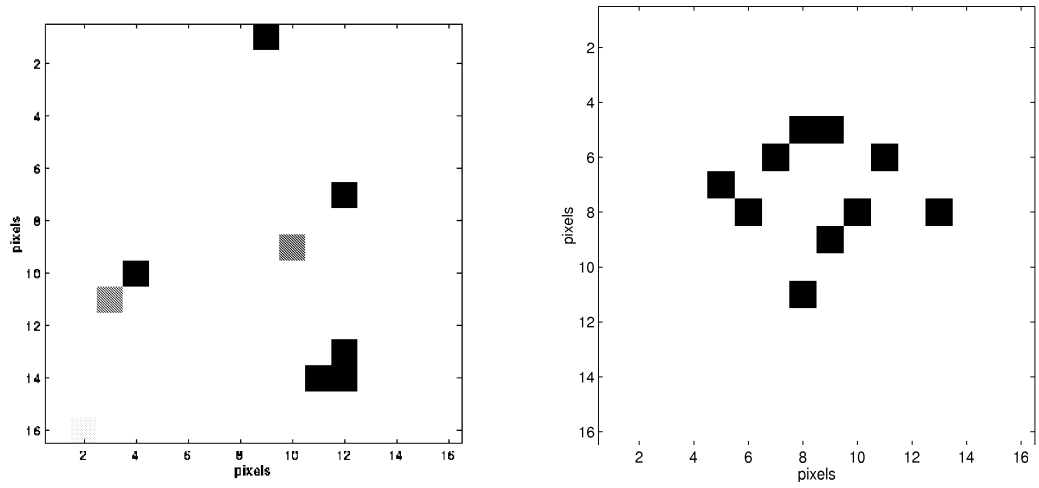
5.7.3 Glass Fibre Impact Damage

Again using the sixteen divide and conquer neural networks but with the addition of using the differencing pre-processing technique (described in Section 4.4.5). The validation data set used was tomo-10, which contained samples of impact damage defects within the second glass fibre material GFRC(b). Figure 5.15(a) illustrates the image formed for the 10J impact damage defect, and indicates an outline of the impact damage. Shown in Figure 5.15(b) is the same image as in Figure 5.15(a) but with the grey scales altered to increase the contrast between 'defect' and 'no defect' outputs. Figure 5.16 compares the images of a 14J impact damage defect and a 18J impact damage and indicates that larger energies tend to cause a greater spread of damage. Note that the exact size and shape of the defects are not known.



(a) original (b) processed

Figure 5.15: Divide and conquer image of the 10J impact damage.



(a) Image of 14J impact (b) Image of 18J impact

Figure 5.16: Comparing the images of the 14J and 18J impact damage defect.

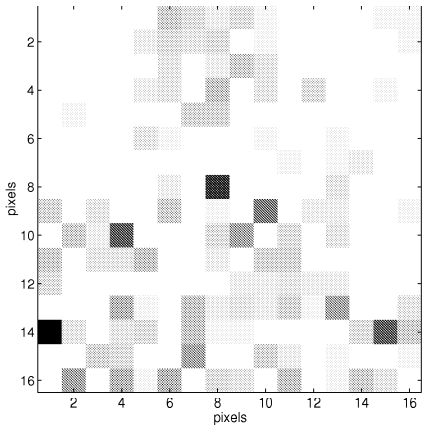
5.7.4 Impact Damage within the Jaguar Wing

The sixteen neural networks have been applied to the Jaguar aircraft wing samples (tomo-11). Figure 5.17(a) and Figure 5.17(b) illustrates the image generated by the neural network system for both of the impact damage defects. With the grey scales altered the images shown in Figure 5.17(c) and Figure 5.17(d) are obtained. Figure 5.17(e) and Figure 5.17(f) illustrates the C-scan images of the two impact defects; these C-scan images are formed by a technique known as the ANDSCAN system, developed by the DRA. Figure 5.17(a) is an image of defect 8, as shown in Figure 4.5, which was formed by a pointed impactor of 1.95kg mass and having 72.4J of energy. Defect 8 has visible surface damage and thus allowed easy location within the centre of the scanning area during data collection. Figure 5.17(c) indicates the outline of the defect (centre) and also there is evidence of a spar positioned below the defect. Defect 9 is imaged in Figure 5.17(d) and was formed by a 1.8kg ball having 63.6J of energy, and is barely visible from the surface.

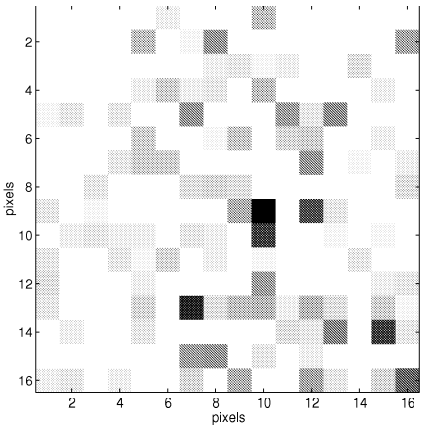
5.7.5 Carbon Fibre Inclusions

Neural networks trained with both the 16 and 40 transducer sensor array data were used with the carbon fibre inclusion validation data (tomo-6 and tomo-9). For the neural network trained with the 16 transducer data, reliable classifications were achieved by using the differencing pre-processing technique. Although none of the defects were detected using the middle threshold bound, good results were given for the smaller ply chopped fibre defects with the carbon fibres aligned with the uni-directional fibres of the training data. Shown in Figure 5.18, the 2 and 6 ply defects are correctly located in the appropriate corner of the scanning area using a decision threshold of 0.8 (defect location and size is indicated by dashed square outline).

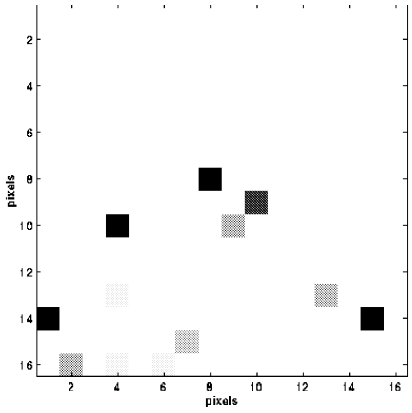
The sixteen divide and conquer neural network system was unable to produce good quality high resolution images with the carbon fibre inclusion defects. There are several possible reasons which may explain such a result. Firstly, the defects are large compared to the 5mm training data used. Secondly, the neural networks were not trained with the carbon fibre, which indicates that to achieve good quality images, the neural network



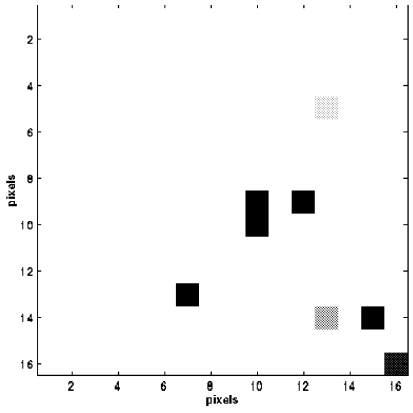
(a) Defect 8



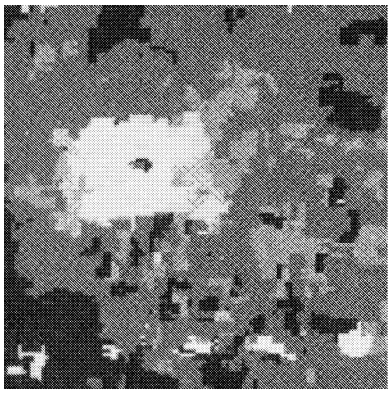
(b) Defect 9



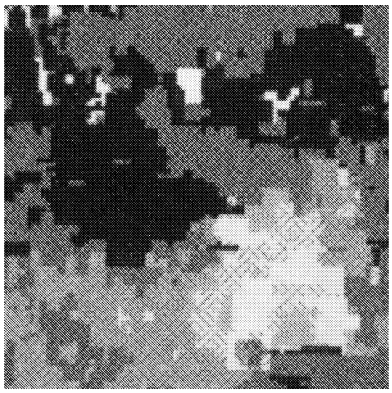
(c) Defect 8



(d) Defect 9



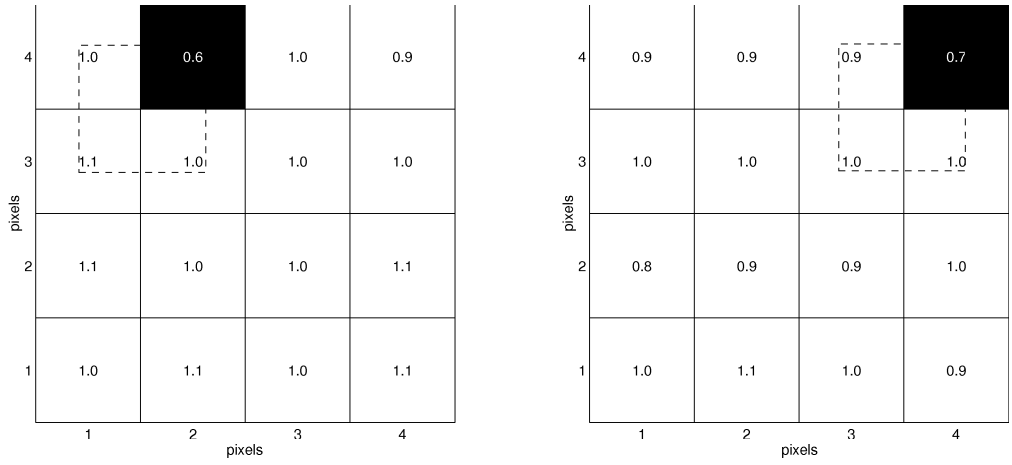
(e) Defect 8



(f) Defect 9

Figure 5.17: Images of the two impact damage defects within the Jaguar wing.

requires to be trained with the same material that will be used after training. And finally, the carbon fibre samples are unidirectional and hence extremely anisotropic, and were expected to generate the lowest quality images. However, even with such a different material to that used to train the neural networks, the low resolution neural network could still generate images which correctly located the defects.



(a) Image of 2 ply

(b) Image of 6 ply

Figure 5.18: Images of the carbon fibre inclusion defects using the 16 transducer sensor array.

5.8 Summary

Validation of the various neural networks demonstrated that the divide and conquer technique, producing 16 by 16 pixel resolution images, worked as expected on a variety of defects including both single and double artificial defects and impact damage in both pultruded material and the Jaguar wing. High resolution imaging was not possible in very anisotropic unidirectional composites, however, low resolution imaging was achieved.

Chapter 6

Neural Networks for C-scan Image Enhancement

6.1 Introduction

In conventional ultrasonic C-scan imaging, a single ultrasonic transducer operating in the pulse-echo mode is moved across the sample in incremental steps. At each increment, when the transducer is motionless, the transducer sends an ultrasonic pulse and captures the received reflections. Typically, the transducer is moved over the sample in two directions forming a zig-zag motion, incrementally, covering a square or rectangular area of the sample. When complete, a series of received ultrasonic waveforms is obtained, one for each of the incremental positions of the scan. Typically, for each waveform, the front and back wall reflections from the sample would be identified and any other reflections in-between may be assumed to be produced from a defect. Having selected a time window for the detection of defect reflections, a single value, for example the peak to peak amplitude of the waveform within the window, can be calculated and used to form an image of the sample.

A common problem associated with C-scan images is that the size of the imaged defect does not exactly correspond to the actual defect size. This is caused by two effects. Firstly, if the width of the transducer's active surface is larger than the scanned defect then the image of the defect will be enlarged by approximately the width of the

transducer. The second effect is the ultrasonic beam divergence of the transducer [3] which will affect all images irrespective of the defect size.

Several methods of image enhancement to reduce errors in sizing defects, due to the effects of transducer size and beam divergence, noise and imperfect fidelity, have been investigated by several researchers and include filtering [111], thresholding [112], histogram equalization, one-dimensional segmentation and intensity scans with hidden line removal [113]. However, they are all computationally intensive and slow. As neural networks have been successfully applied to other image enhancement problems [114, 115], a proposed neural network method of enhancing original C-scan images to produce images which better illustrate the actual size of any defect imaged is investigated.

This chapter is divided into five sections. The first details the data collection procedure and the algorithms employed to perform data pre-processing and target generation. The next section describes how neural network training, testing and validation data sets were formed from the collected data. This is followed by an initial investigation of neural networks applied to simple line-scan data. Then, a neural network system developed to enhance C-scan images is detailed. Finally, the neural network system is validated with additional C-scan images.

6.2 Data Collection and Pre-processing

The collected data consists of C-scan images with a resolution of 40 by 40 pixels, where one pixel represents a movement of the ultrasonic transducer of 1mm. A Panametrics immersion transducer with a 3.5MHz bandwidth and 10mm active surface diameter was used. The transducer is held in a fixed vertical position above a water tank. The instrumentation consists of Panametrics 5055PR signal pulser attached to the transducer and operating in the pulse-echo mode, the received signal of which is directed to a Tektronix 2430A digital oscilloscope. A water tank of approximate size 400mm by 300mm by 80mm is placed on top of a Naples Coombe[1] XY flat bed scanner which is controlled by a Minicam[2] stepper motor controller. Both the Tektronix oscilloscope and the Minicam motor controller are connected to a IBM computer via the GPIB IEEE 488 interface bus. The computer has instrument control software, to control the operation of the digital oscilloscope and the XY scanner, which automates the acquisition of C-scan images.

A Perspex block, dimensions 300 x 200 x 12.7mm, contained ten different defects, in the form of flat-bottomed circular holes, with diameters of 2, 3, 4, 5, 6, 8, 10, 12, 15 and 20mm. The holes were drilled from the back surface of the block to a depth of 5mm. For each defect, seven C-scan images were collected, giving a database of 70 C-scan images. Figure 6.1 illustrates the C-scan images for one of the seven examples of each defect size.[3] Note that for each example the defect was positioned at approximately the centre of the imaging area, however, the exact position is unknown.

Additional validation data was also collected using square defects cut into another perspex block, again the block was 12.7mm thick. Six defects were formed and each had a depth of 5mm and included widths of 4, 6, 8, 10, 12 and 20mm. Figure 6.2 illustrates the C-scan images of these six square defects.

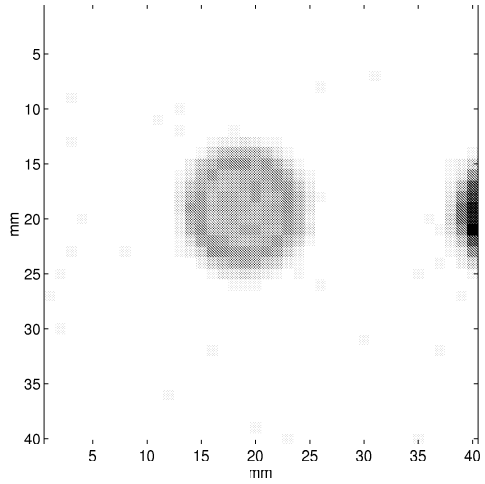
Two pre-processing methods have been used on the C-scan data. The first method generated a series of line-scans from each C-scan image. A line-scan image can be

[1] Naples Coombes Ltd., Chaddleworth, Berkshire, UK.
[2] Bede Scientific Instruments Ltd., Coxhoe, Co. Durham, UK.
[3] The colour maps for these images are scaled so that the smallest value in the image is represented by white and the largest value by black.

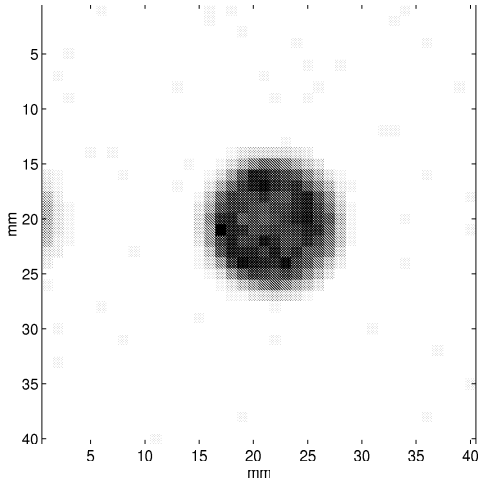
thought of as a single slice of a C-scan image, either a horizontal or vertical slice where the transducer is only moved in the one direction. For a given C-scan image the selection of line-scan slices is determined by the algorithm described as pseudo code in Table 6.1 which selects only horizontal and vertical slices that overlap the centre of the defect as shown in Figure 6.3.

The second pre-processing method is performed in two parts. The first part is to determine the target position of the defect from the original C-scan image of the defect. The algorithm given as pseudo code in Table 6.2 describes how the centre of the defect's position is estimated from the original image, allowing a target image of the defect to be generated as the actual size and shape are known. Figure 6.4 shows how the C-scan images compare with the estimated target images.

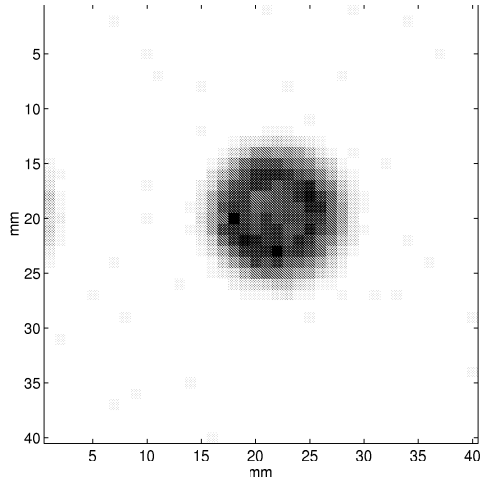
The second part is required because of the limited number of C-scan images. This limitation prevents the enhancement of the 40 by 40 pixel image directly. Instead a neural network is to be trained with a 10 by 10 pixel sub-image of the original C-scan image. The intention is to have a neural network which may take any 10 by 10 sub-image of the C-scan image and produce as output an enhanced 10 by 10 pixel image. Thus, by passing the neural network "completely over" the C-scan image, an enhanced 40 by 40 pixel image will be produced. In practice, for a given C-scan image, sub-images are generated such that the centre of the target defect image is passed through the 10 by 10 pixel image, as described by the algorithm given as pseudo code in Table 6.3. Figure 6.5 illustrates the initial position of the sub-image window on the original C-scan image and its target image, while Figure 6.6 gives a few examples of sub-images generated from one original C-scan image.



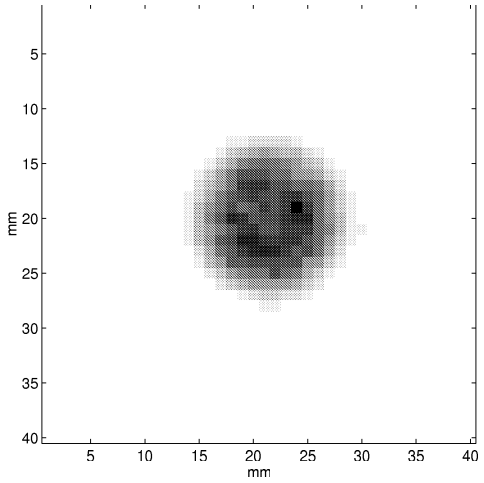
(a) 2mm diameter



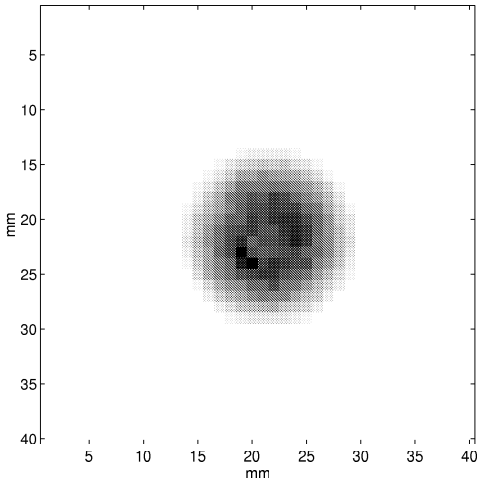
(b) 3mm diameter



(c) 4mm diameter

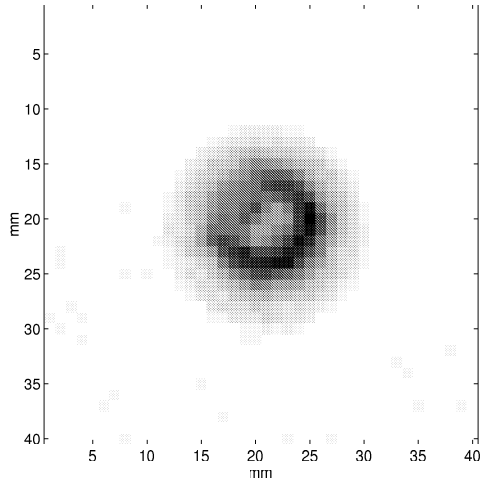


(d) 5mm diameter

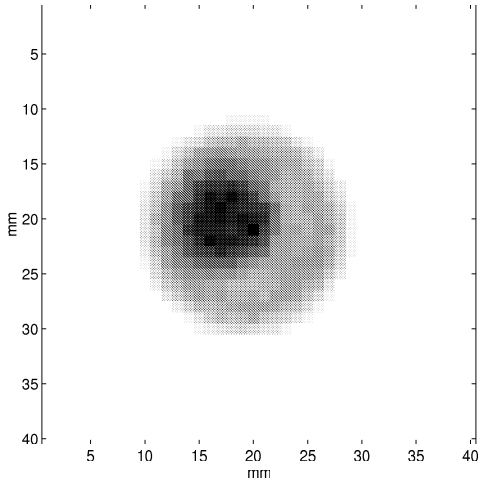


(e) 6mm diameter

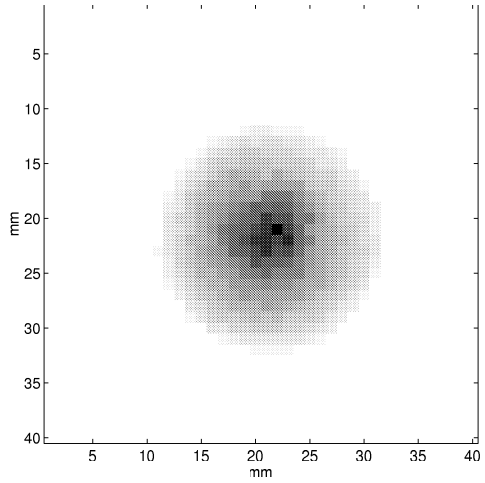
Figure 6.1: C-scan images of circular holes of 2 to 6mm diameters.



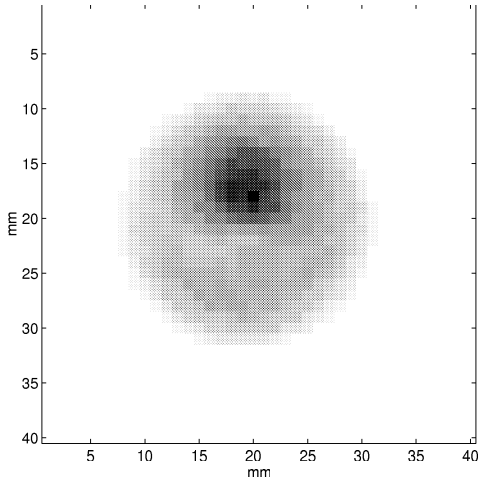
(f) 8mm diameter



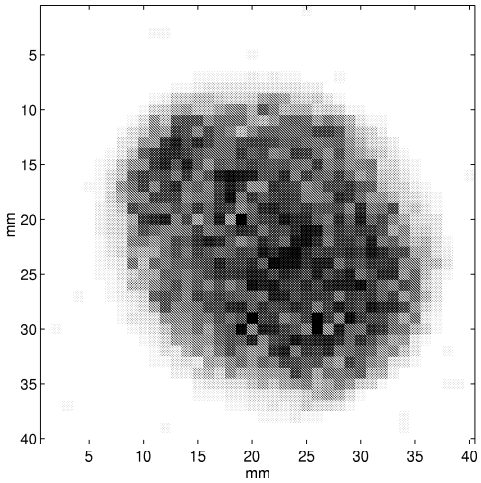
(g) 10mm diameter



(h) 12mm diameter

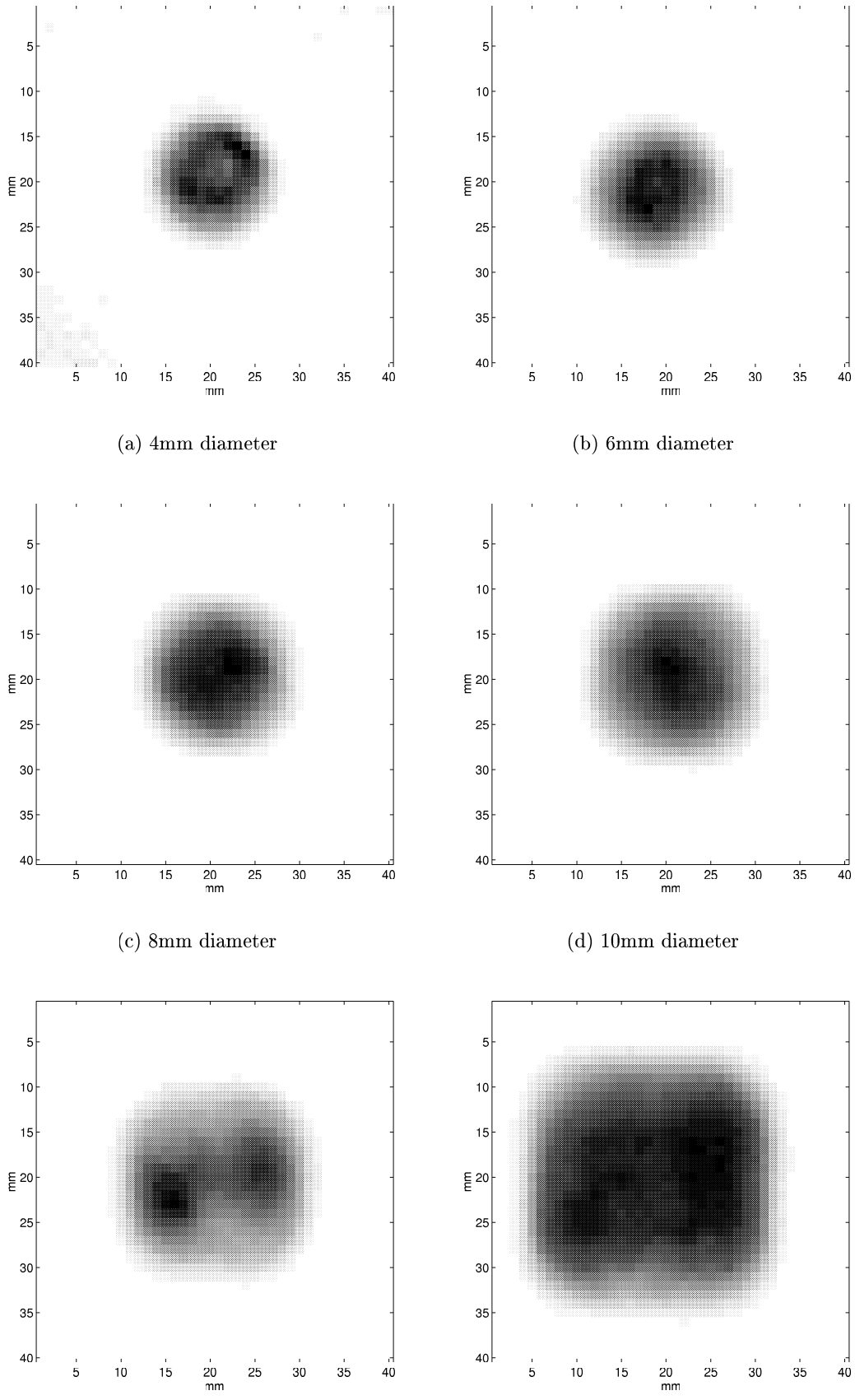


(i) 15mm diameter



(j) 20mm diameter

Figure 6.1: C-scan images of circular holes of 8, 10, 12, 15 and 20mm diameters.



(a) 4mm diameter

(b) 6mm diameter

(c) 8mm diameter

(d) 10mm diameter

(e) 12mm diameter

(f) 20mm diameter

Figure 6.2: C-scan images of square hole defects.

Table 6.1: Selection of Line-scan slices from C-scan image.

Require: one C-scan image of known defect size
Ensure: $0 < imagefraction \leq 1$
  defectsize ← the actual size of the defect
  maximageval ← largest numerical value in c-scan image
  for each corner of the c-scan image **do**
    selectpixel = corner pixel
    while value of *selectpixel* < *imagefraction* × *maximageval* **do**
      let *selectpixel* point to the next pixel in the image
    end while
    cornerpixel ← *selectpixel*
  end for
  use the rectangle defined by *cornerpixel*'s to select the horizontal and vertical Line-scan slices

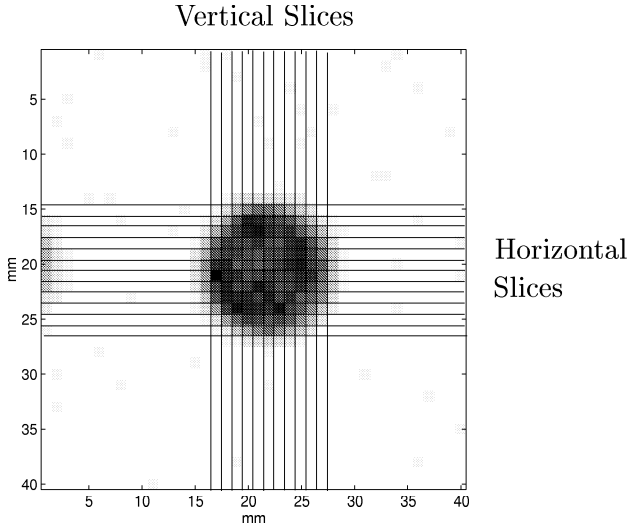


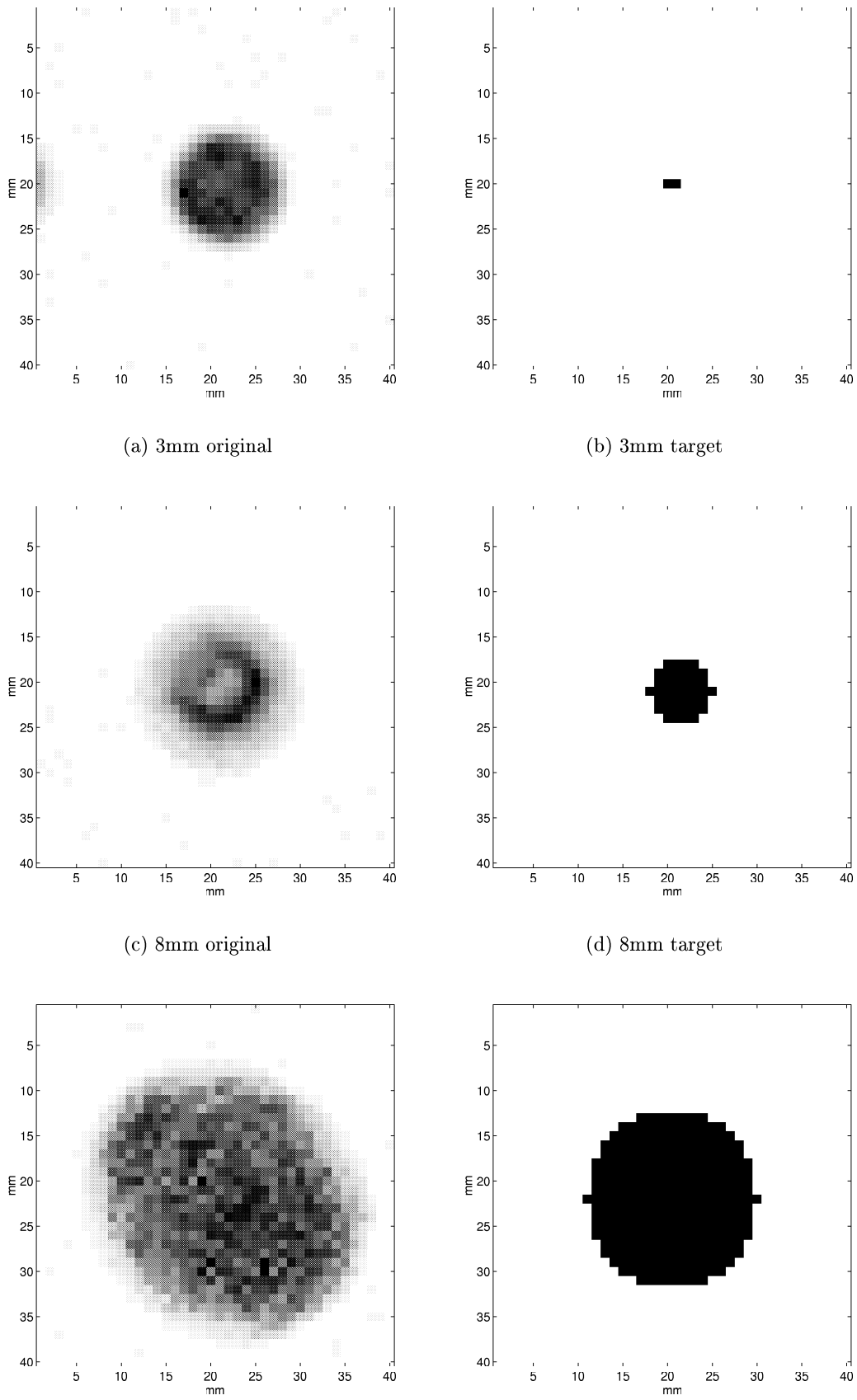
Figure 6.3: Illustration of Line-scan slices taken from C-scan image (*imagefraction* = 0.5).

Table 6.2: Generate a target image from an original C-scan image.

Require: one C-scan image of known defect size
Ensure: $0 < imagefraction \leq 1$
  defectsize ← the actual size of the defect
  maximageval ← largest numerical value in c-scan image
  for each corner of the c-scan image **do**
    selectpixel = corner pixel
    while value of *selectpixel* < *imagefraction* × *maximageval* **do**
      let *selectpixel* point to the next pixel in the image
    end while
    cornerpixel ← *selectpixel*
  end for
  defectcen ← centre of rectangle defined by *cornerpixel*'s
    {estimate of defect centre}
  output target image of a circular defect, centred at *defectcen*
    and of size *defectsize*

Table 6.3: Pass image through a 10 by 10 pixel sub-image window.

Require: one C-scan image and it's associated target image
Ensure: *defectcen* is known {defect centre}
  take a 10 by 10 pixel sub-image window and place
    the bottom right hand corner over *defectcen*
  while *defectcen* has not been positioned
    at every pixel within the window **do**
    output the c-scan image contained within the 10 by 10 pixel window
    also output the target image contained within the window
    shift the window by one pixel
  end while



(a) 3mm original

(b) 3mm target

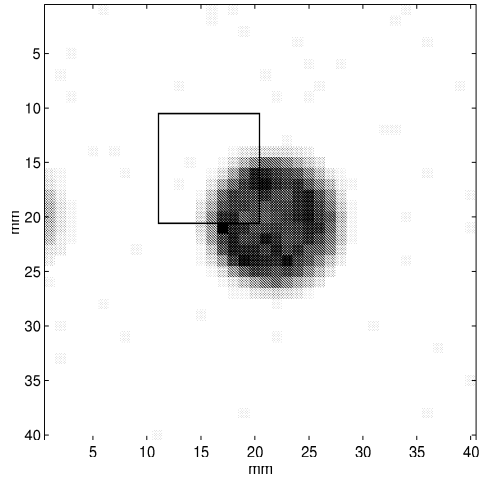
(c) 8mm original

(d) 8mm target

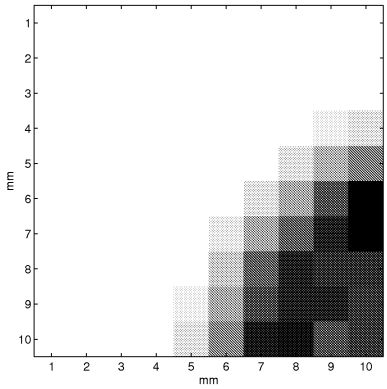
(e) 20mm original

(f) 20mm target

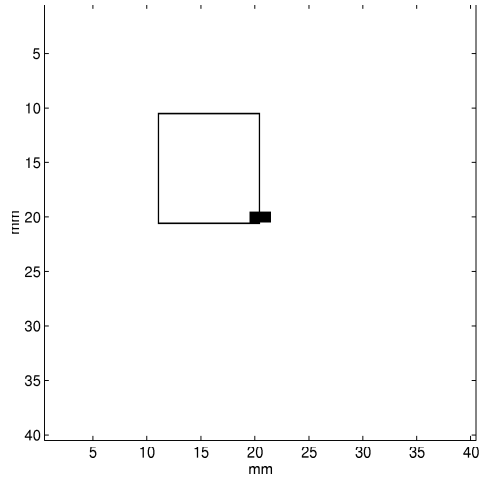
Figure 6.4: Original c-scan image and its associated target image.



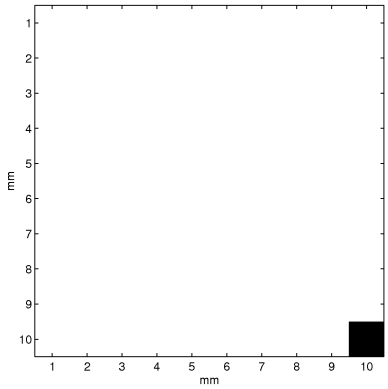
(a) Original



(b) Original sub-image

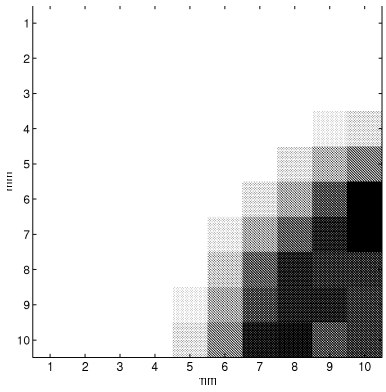


(c) Target

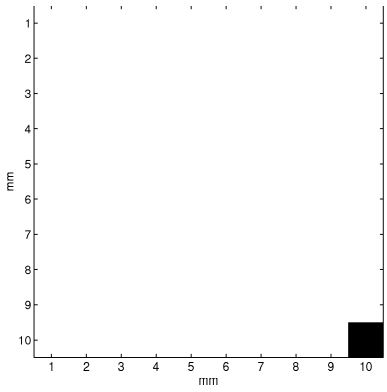


(d) Target sub-image

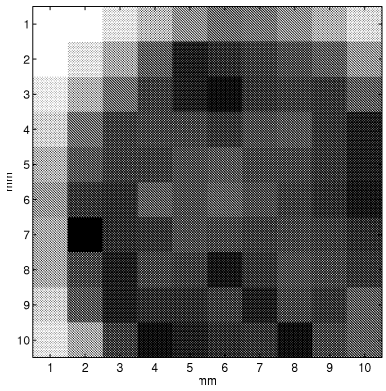
Figure 6.5: Sub-images of the original 3mm defect.



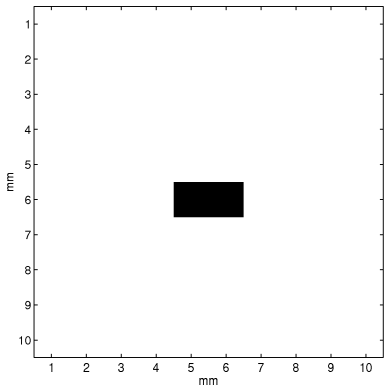
(a) Sub-image



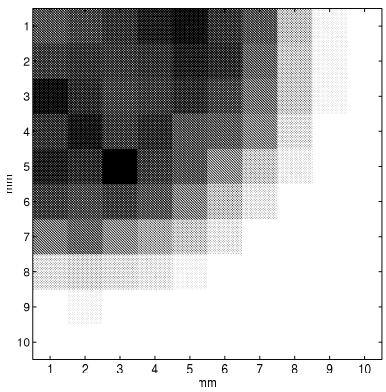
(b) Target



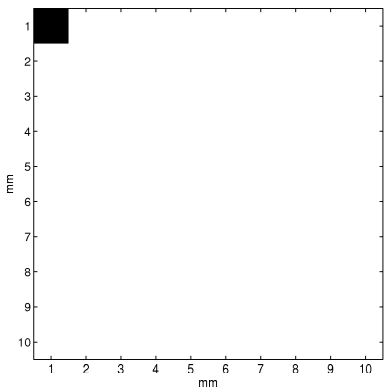
(c) Sub-image



(d) Target



(e) Sub-image



(f) Target

Figure 6.6: Other examples of sub-images and targets for the 3mm defect.

6.3 The Data Sets

The data sets are divided into two categories relating to the two different pre-processing methods previously described. Note that the original C-scan images are divided into seven groups of ten, thus each group has one example of every defect size. The first category concerns the generation of the Line-scan images from the C-scan images. Three of the seven groups of ten C-scan images are used to produce three data sets of horizontal line-scan slices and three data sets of vertical line-scan slices. Two of the three data sets will be used as training data while the third forms the testing data. Each data set contains examples of all ten different defect sizes. The data sets are given target values which represent the actual width of the defect, the target coding methods used are detailed in Section 6.4.1.

The second category produces sub-image data. The seven groups of original C-scan images are pre-processed separately to give seven data sets, which may be used for either training, testing or validation data. Note that the pre-processing technique allows the number of patterns in each group to increase from the original 10 images to approximately 2,000 sub-images. The data sets contain examples of both the C-scan sub-image and the target sub-image. Note that the target sub-images are encoded into the numerical range of -1 to 1, where -1 represents a 'no defect' and 1 indicates a 'defect' at the specified pixel.

Before being presented to the neural networks, each data set is re-scaled with respect to each input (image pixel) into the range of -1 to 1. As previously discussed this helps the learning process of the neural network. Note that the actual range of values within the C-scan images are typically between 0 and 3.2 and appear to be correlated to the defect size. The larger defects will give larger pixel values because the reflected ultrasonic signal from the defect will be maximised when the diameter of the defect is equal or greater to the diameter of the transducer.

6.4 Initial Studies with the Line-scan Data

Before using a neural network to perform image enhancement of C-scan images, line-scan data is used to assess the neural network's ability to determine the actual width of a defect from line-scan slices taken from C-scan images. Before detailing the results, the various methods of encoding the target output of width are described.

6.4.1 Target Encoding of Width

There are several methods available to encode the width of the defect. The simplest method will encode the various sizes, via a re-scale, into the numerical range of either 0 to 1 or -1 to 1, thus allowing a single output neuron to indicate all possible sizes. However, this type of encoding produces undesirably small numerical regions for the uncertain classifications between two training sizes.

Another method is to have several output neurons, each one operating as a flag for a certain size. Hence, if 6 different defect widths were present in the training data then six output neurons would be used, each acting as a binary flag for a different defect size. Such a method limits the generalization ability to identify unseen sizes.

This leads to the third method which uses several output neurons to encode some form of counting system covering the numerical range of the defect sizes within the training data. Four numerical encoding techniques have been used and included binary, grey, spread and thermometer coding. For all four encoding methods, any given output may be either inactive or active. Here inactive is represented by -1 and active by 1. Note that grey encoding is similar to binary except that only one bit may change between two consecutive numbers and spread encoding is similar to thermometer encoding except that only the bit representing the number is set to active (with thermometer encoding all the numerically lower bits are also active). These four techniques were compared and the performance of each is given in Table 6.4. As indicated, spread and thermometer encoding produce acceptable performance (approaching or above 90% correct).

6.4.2 Classification of Defect Width

Initial neural network optimization gave a learning rate of 0.05, a momentum of 0.9 and 10 hidden neurons. The network architecture was 40 inputs with 10 output neurons. The standard BP algorithm was used with continuous updating of weights and the bipolar activation function. The training patterns were randomly presented to the neural network during training. Note that although the smaller defects had fewer slices, and hence fewer training patterns, duplication of training data to produce evenly represented classes was not implemented.

Table 6.5 shows the testing results of four different neural networks. Combinations of horizontal and vertical slices and spread and thermometer encoding are used to form the four different neural network training and testing data. The line-scan slides used for these NN were generated with an image fraction value of 0.5. As shown the thermometer encoding performs slightly better than the spread encoding. The vertical slices matched with the thermometer encoding gives the overall highest performance.

By means of comparison, another set of four neural networks have been trained with data generated with an image fraction of 0.8. Shown in Table 6.6 the results of all four neural networks have been improved. This result was expected because increasing the image fraction effectively reduces the number of slices assumed to be centrally located over a defect. And although this reduces the amount of available data, the selected slices are more consistent with a middle cross section of a defect.

Examination of the mis-classified examples from the testing data of the vertical slices using thermometer encoding and an image fraction of 0.8, indicate that, 78% of the mis-classified sizes were only one size away from the correct value and 10% were only two sizes away from the correct size.

Table 6.4: Comparing target encoding methods.

Target Encoding	% Corr
Binary	60
Grey	75
Spread	87
Thermometer	91

Table 6.5: Line-scan width classification with an image fraction of 0.5.

Line-scan slice type	Target Encoding	Performance Metrics				
		Sens	Spec	PosPV	FalAR	% Corr
Horizontal	Spread	0.27	0.92	0.29	0.07	86.0
	Thermometer	0.95	0.76	0.87	0.23	88.2
Vertical	Spread	0.33	0.93	0.35	0.06	87.4
	Thermometer	0.94	0.86	0.91	0.13	91.3

Table 6.6: Line-scan width classification with an image fraction of 0.8.

Line-scan slice type	Target Encoding	Performance Metrics				
		Sens	Spec	PosPV	FalAR	% Corr
Horizontal	Spread	0.18	0.94	0.29	0.05	87.2
	Thermometer	0.92	0.88	0.93	0.11	91.41
Vertical	Spread	0.23	0.95	0.39	0.04	88.7
	Thermometer	0.90	0.93	0.94	0.06	91.9

6.5 Neural Networks Enhancement of C-scan Images

Application of equation 2.41 suggests that the amount of training data currently available prevents the development of a neural network which can directly enhance 40 by 40 pixel resolution images. The solution to this limitation is to firstly develop a neural network which can enhance small portions, called sub-images, of a complete C-scan image. Then by dividing the original C-scan image into several sub-images, presenting them sequentially to the neural network and combining the resulting enhanced sub-images, a complete enhanced image can be formed.

Preliminary results are given in Section 6.5.1 for a neural network which enhances a 10 by 10 pixel sub-image extracted from the original C-scan images. This is followed by details of several schemes employed to enhance the complete 40 by 40 pixel images and includes a comparison of the performance with the validation data.

6.5.1 Sub-image Enhancement

All of the sub-image enhancement results are from a neural network using the standard BP algorithm with continuous updating and the bipolar activation function. The training patterns were randomly presented to the neural network during training. In addition to the usual neural network optimization parameters, the image fraction variable also requires optimization. Thus optimization of the neural networks included a search of the image fraction parameter over the range 0.5 to 0.9, learning rates ranged from 0.1 to 0.001, momentum values from 0.5 to 0.9 and the number of hidden neurons from 40 to 90 (number of inputs and outputs both set at 100). The optimum parameter values found were, learning rate 0.05, momentum 0.9, number of hidden neurons to be 80 and an image fraction of 0.9.

Having performed the neural network optimization, the first investigation is to compare the performance of the seven different data sets. Table 6.7 illustrates the testing results of seven neural networks, each one trained with a different data set. As can be seen the neural network trained with data set 5 gives the highest percentage correct value when tested. Detailed performance metrics for this neural network are given in Table 6.8.

Table 6.7: Comparing the selection of the training data.

Data Sets		Performance Metrics				
Training	Testing	Sens	Spec	PosPV	FalAR	% Corr
1	2	0.74	0.95	0.78	0.046	91.7
2	3	0.74	0.96	0.83	0.04	91.8
3	4	0.64	0.97	0.87	0.03	88.0
4	5	0.81	0.95	0.86	0.05	91.4
5	6	0.87	0.94	0.84	0.06	92.3
6	7	0.86	0.90	0.78	0.10	88.7
7	1	0.77	0.90	0.69	0.10	86.9

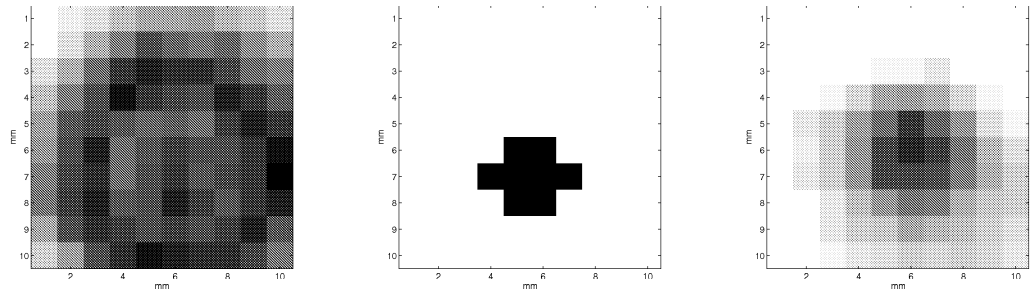
Table 6.8: Performance metrics for the neural network trained with data set 5, tested with data set 6.

	Threshold				
Measure	-0.4	-0.2	0.0	0.2	0.4
TP	50350	48694	46638	44115	40928
TN	135141	139357	142543	145024	146962
FP	16051	11835	8649	6168	4230
FN	3358	5014	7070	9593	12780
Sens	0.94	0.91	0.87	0.82	0.76
Spec	0.89	0.92	0.94	0.96	0.97
PosPV	0.76	0.80	0.84	0.88	0.91
FalAR	0.11	0.08	0.06	0.04	0.03
% Corr	90.53	91.78	92.33	92.31	91.70

The neural network trained with data set 5 was validated with data sets, 1, 2, 3, 4, and 7. The average results give a percentage correct value of 91.7%, a sensitivity of 0.84, a specificity of 0.94 and a positive predictive value of 0.79. Although these results indicate good overall performance it is difficult to appreciate the images generated by the neural network, and so example images, taken from validation data set 3, are given in Figure 6.7 to Figure 6.9. Illustrated are the original sub-images presented to the neural network as input[4], the actual size and location of the defect is demonstrated in the target image and the enhanced image shows the enhanced sub-image generated by the neural network[5]. The example images given, illustrate the smaller defects located within the middle of the 10 by 10 pixel sub-image, however, similar enhancement performance is achieved with defects, of any size, positioned anywhere within the sub-image, however, for the smallest defect size the enhanced image is limited by the resolution ability.

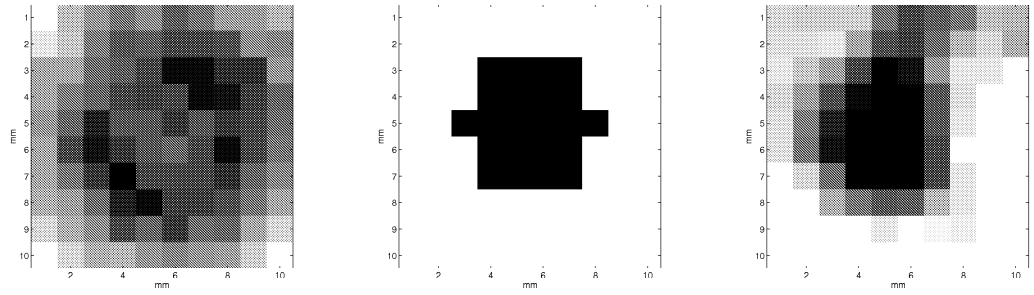
[4]To aid visualization the colour maps for these images are scaled to fit the range of values within the sub-image being displayed. Although the actual sub-images presented to the neural network will be re-scaled with respect to the complete data set. The result of which is that the original sub-images given in Figure 6.7 to Figure 6.9 have greater contrast than the actual sub-images presented to the neural network.

[5]The colour maps for these images are scale between -1 and 1.



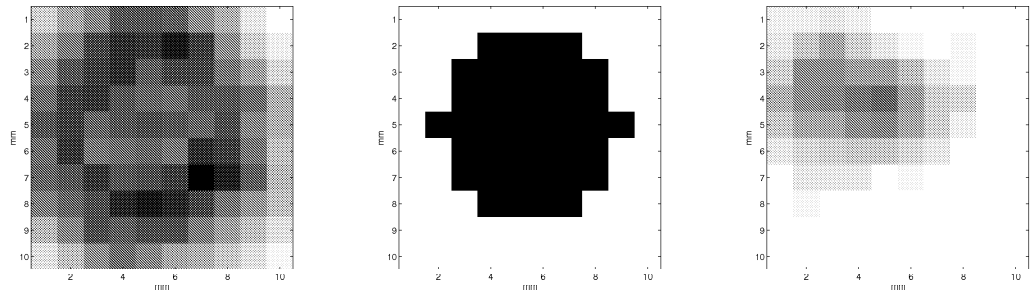
(a) Original (b) Target (c) Enhanced

Figure 6.7: Example of enhanced sub-image of a 4mm defect with its original and target sub-images.



(a) Original (b) Target (c) Enhanced

Figure 6.8: Example of enhanced sub-image of a 6mm defect with its original and target sub-images.



(a) Original (b) Target (c) Enhanced

Figure 6.9: Example of enhanced sub-image of a 8mm defect with its original and target sub-images

6.5.2 Methods for Complete Image Enhancement

Described in the previous section, a neural network was developed which could generate a 10 by 10 pixel enhanced sub-image taken from any portion of the original 40 by 40 pixel C-scan image. For enhancement of the complete C-scan image a system which repeatedly applies this neural network to different portions of the original C-scan image was required. A major consideration when doing this was how to divide the original C-scan image into 10 by 10 pixel sub-image portions. For example, with the original C-scan image of 40 by 40 pixels, should the system simply divide the image area into sixteen unique 10 by 10 pixel sub-image portions or should there be some form of overlap of the portions where overlapped areas are averaged to produce the enhanced image.

Initially the original C-scan images were enhanced by a system which used sixteen unique portions of the complete image, arranged as a 4 by 4 array of 10 by 10 pixel portions or sub-images. Again using the neural network detailed in the previous section which was trained with data set 5 and tested with data set 6. Images of all defect sizes are given in Figure 6.10 to Figure 6.19 for validation data set 3. The figures illustrate both the original C-scan image and the enhanced image generated by the neural network[6]. All enhanced images, for sizes above 5mm, are an improvement on the originals in terms of defect size.

Although the system worked well with just sixteen unique portions of the complete C-scan image, a system with overlapping portions was investigated. Here, in addition to the previous sixteen portions another nine portions are sampled. These nine portions are arranged in a 3 by 3 array and offset from the edges of the original image by 5 pixels in both directions, such that the centre of any of the nine portions overlap the point where four corners from adjacent portions of the sixteen meet. Figure 6.20 and Figure 6.21 illustrate both the enhanced image produced from the sixteen and the nine portions[7] together with the original C-scan image and the combined or average

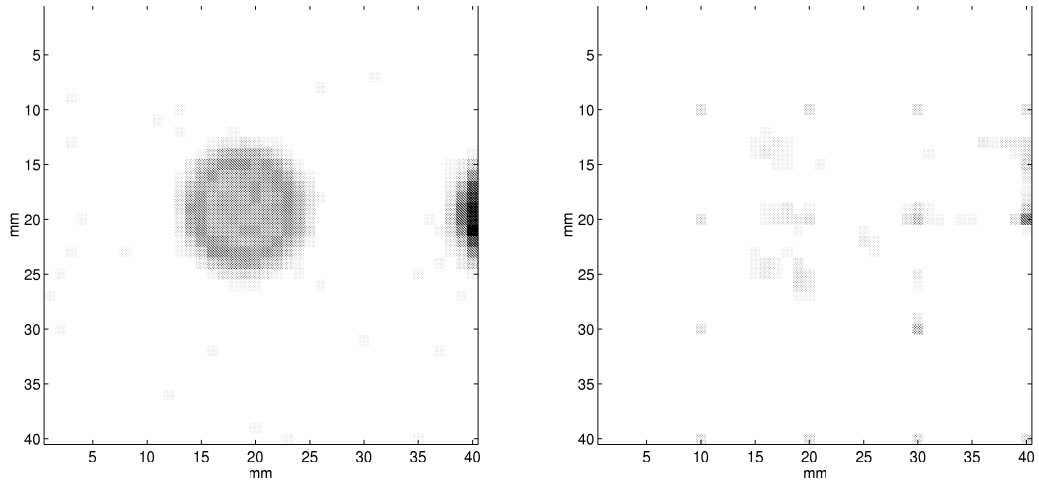
[6]The original images colour maps are scaled as before to fit the variation in values within the image while the enhanced images have colour maps scaled to the range of -1 to 1

[7]The enhanced images shown for the nine portion's have a 5 pixel wide border around the edge, indicating the area of the original C-scan image that is not enhanced. When the combined average enhancement image is formed this border is set to the minimum value of -1.

enhancement, for the 3mm and 20mm diameter defects. Except for the smallest (2mm) defect, all of the combined enhancement images give visible improvements, although the most noticeable improvement is with the 20mm defect in which the combined enhancement gives a significantly closer representation of a circular defect.

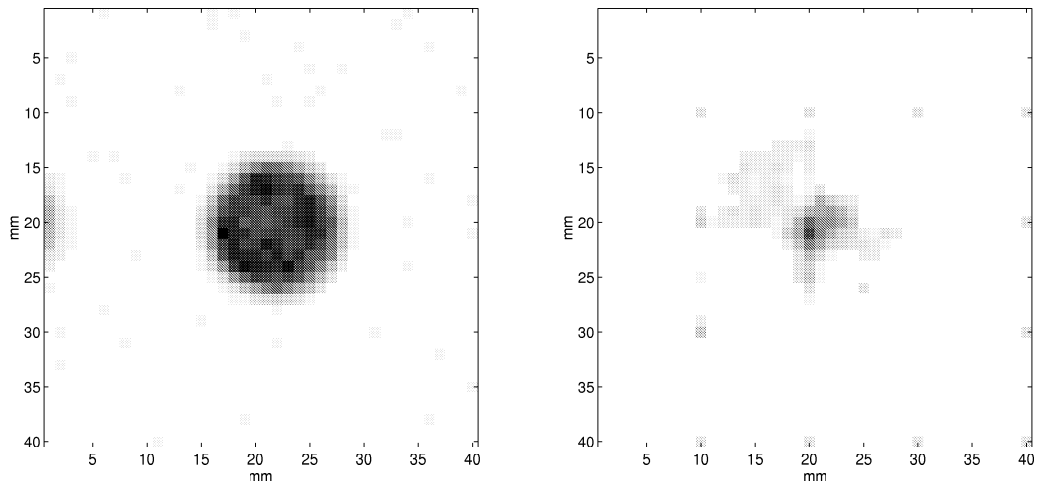
The system which combines the sixteen and nine portion enhanced images has been used on the square defect validation data set. The resulting images for the 4mm and 20mm defects are shown in Figure 6.22 and Figure 6.23. Although the neural network was only trained with circular defects the enhancement of the square defects is extremely good, with a definite visual suggestion of squareness for the larger sized defects.

The system of incorporating overlapping portions of the complete C-scan image may be extended to its maximum potential. Here the 10 by 10 pixel sub-image neural network is scanned over the complete original C-scan image one pixel at a time until the sub-image has been positioned at every possible location within the complete image. By averaging all of the values generated, for a given complete image pixel, by the numerous sub-images which specify that pixel, a 'pixel by pixel' enhancement image can be formed. Figure 6.24 and Figure 6.25 give the enhancement images formed for the 20mm circular and square defects. Due to the large number of values averaged per pixel the resulting enhancement images are comparatively smooth.



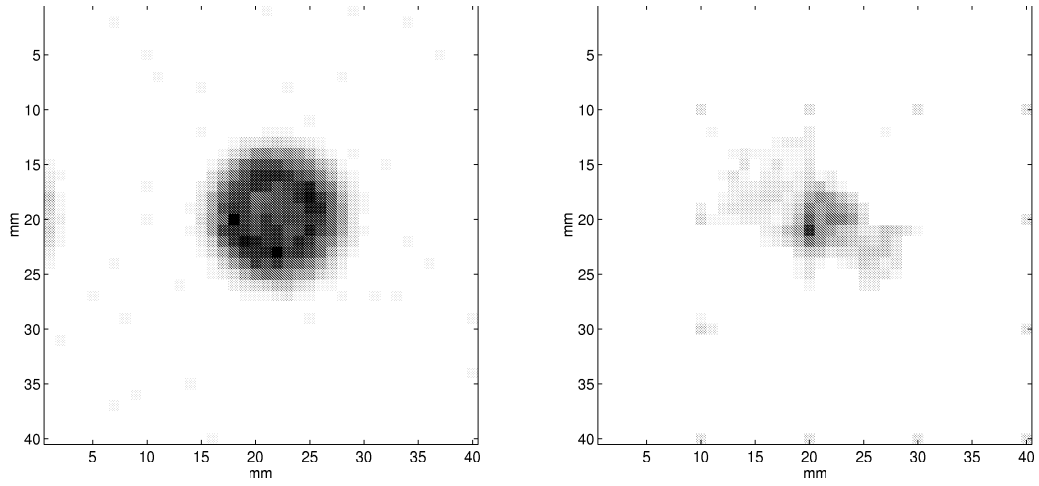
(a) Original (b) Enhanced

Figure 6.10: 2mm defect original C-scan and sixteen portion enhanced image.



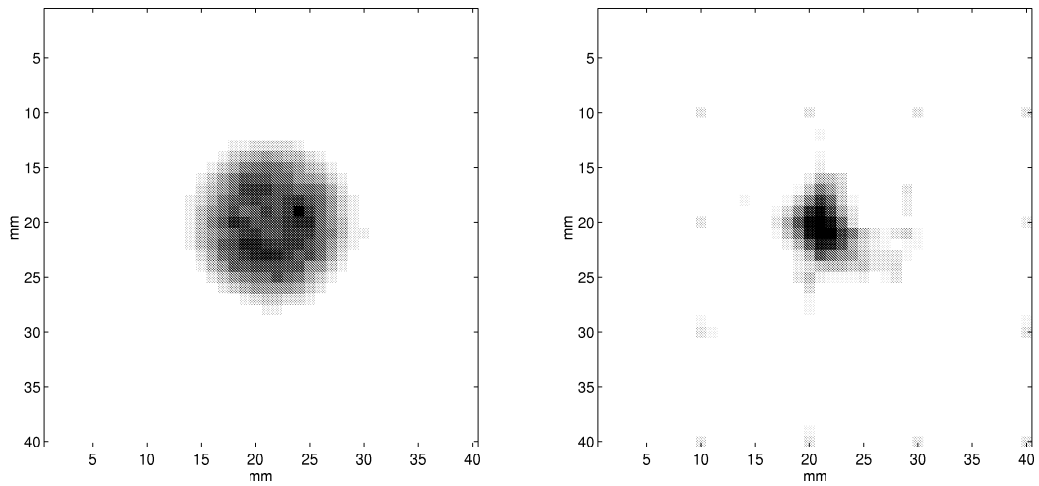
(a) Original (b) Enhanced

Figure 6.11: 3mm defect original C-scan and sixteen portion enhanced image.



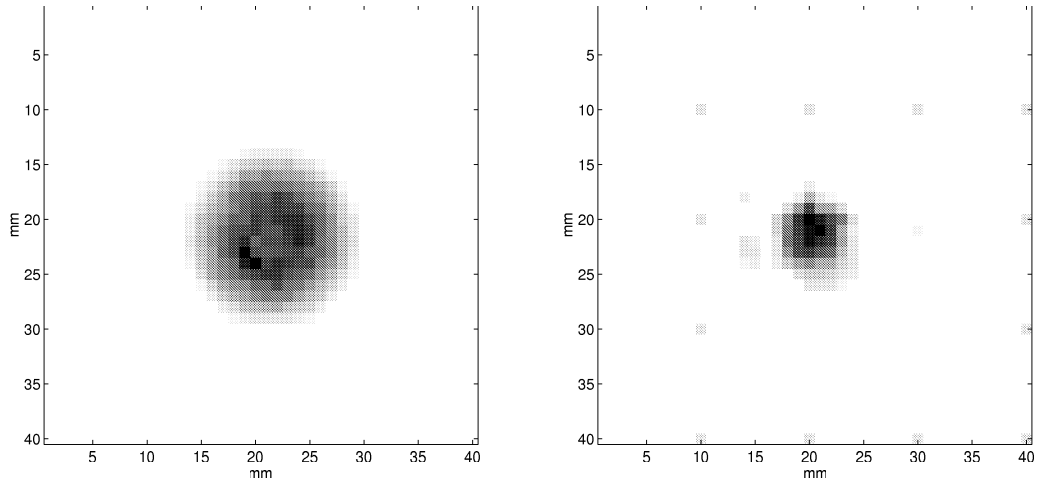
(a) Original (b) Enhanced

Figure 6.12: 4mm defect original C-scan and sixteen portion enhanced image.



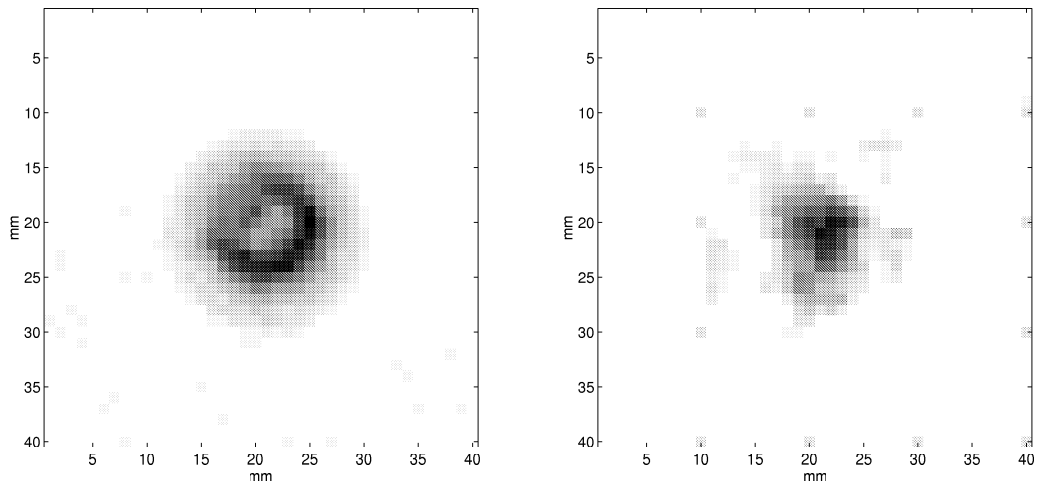
(a) Original (b) Enhanced

Figure 6.13: 5mm defect original C-scan and sixteen portion enhanced image.



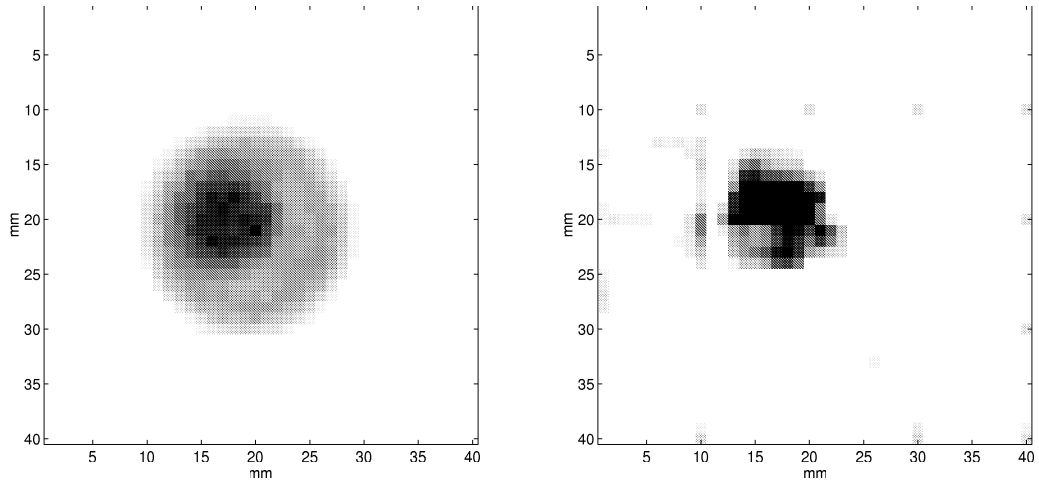
(a) Original (b) Enhanced

Figure 6.14: 6mm defect original C-scan and sixteen portion enhanced image.



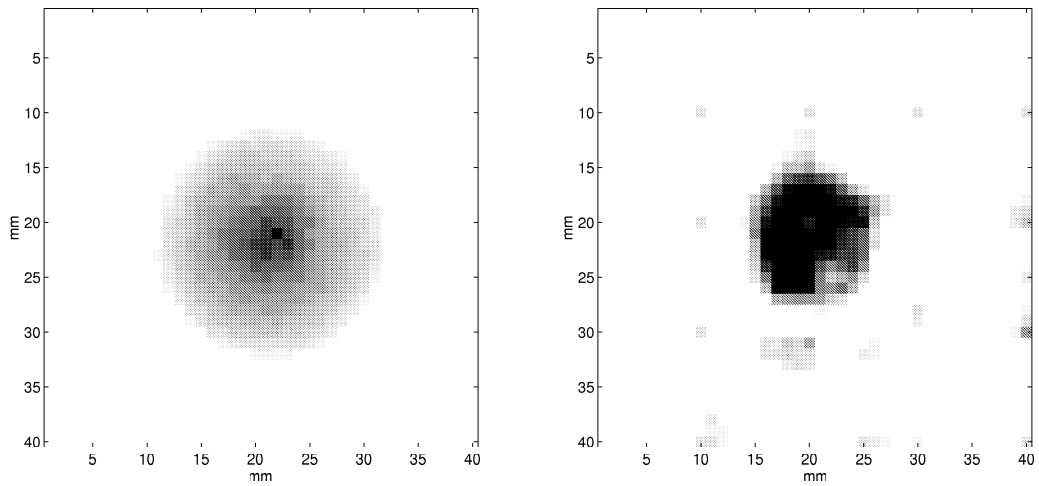
(a) Original (b) Enhanced

Figure 6.15: 8mm defect original C-scan and sixteen portion enhanced image.



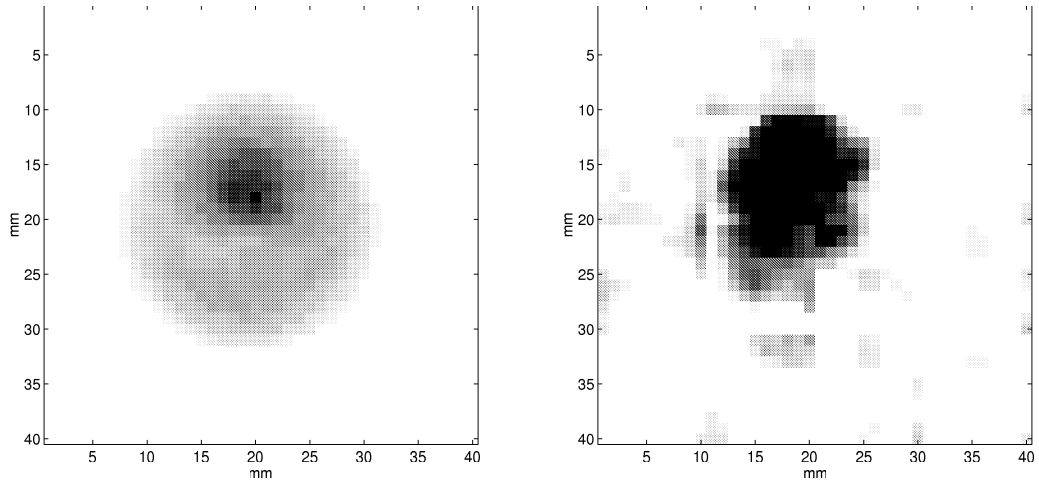
(a) Original (b) Enhanced

Figure 6.16: 10mm defect original C-scan and sixteen portion enhanced image.



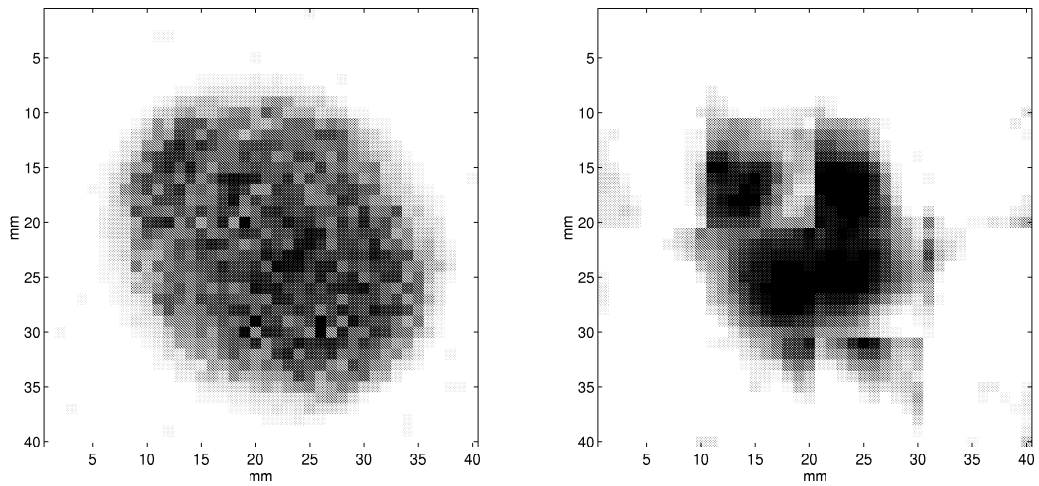
(a) Original (b) Enhanced

Figure 6.17: 12mm defect original C-scan and sixteen portion enhanced image.



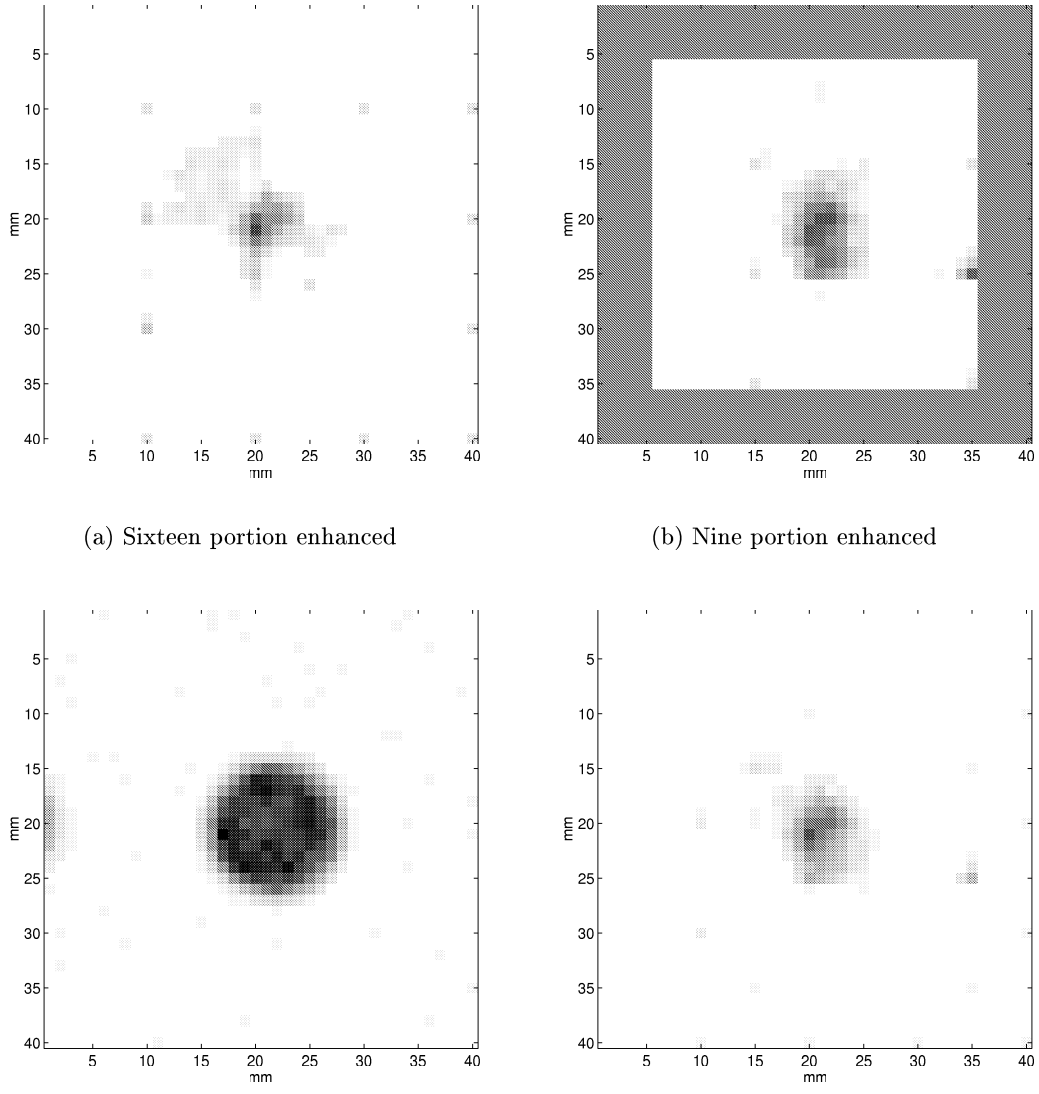
(a) Original (b) Enhanced

Figure 6.18: 15mm defect original C-scan and sixteen portion enhanced image.



(a) Original (b) Enhanced

Figure 6.19: 20mm defect original C-scan and sixteen portion enhanced image.



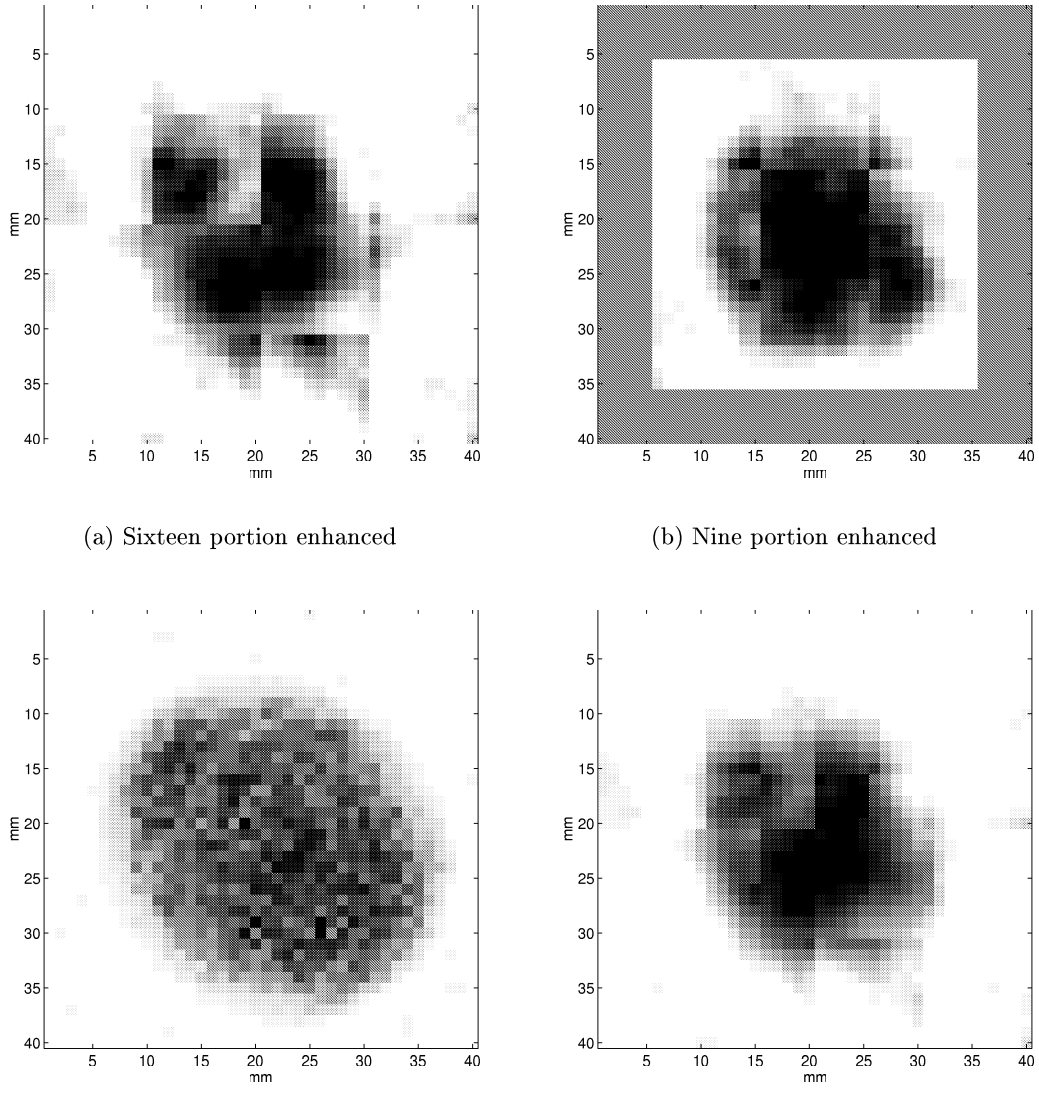
(a) Sixteen portion enhanced

(b) Nine portion enhanced

(c) Original

(d) Combined enhancement

Figure 6.20: 3mm defect, sixteen and nine portion and combined enhancement.



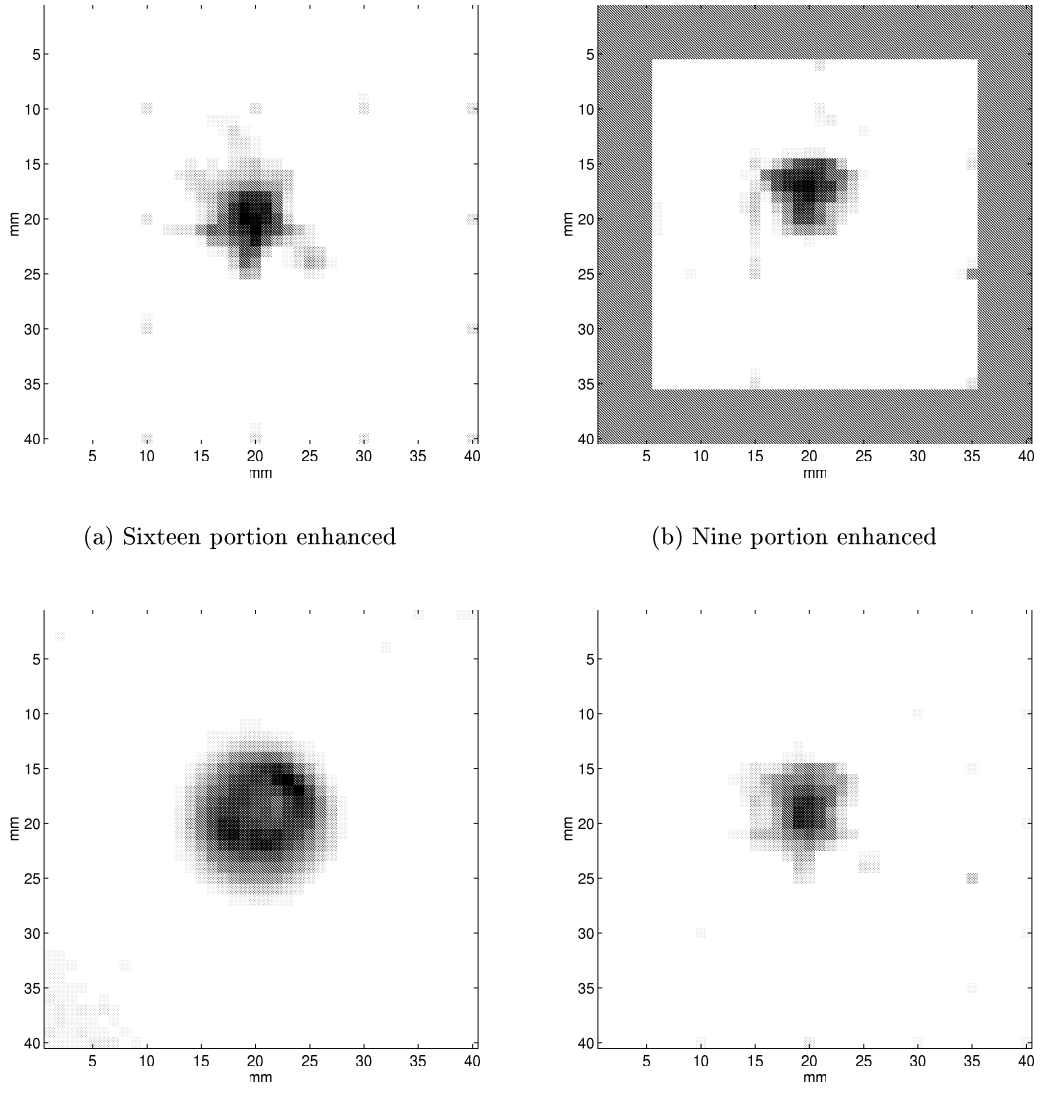
(a) Sixteen portion enhanced

(b) Nine portion enhanced

(c) Original

(d) Combined enhancement

Figure 6.21: 20mm defect, sixteen and nine portion and combined enhancement.



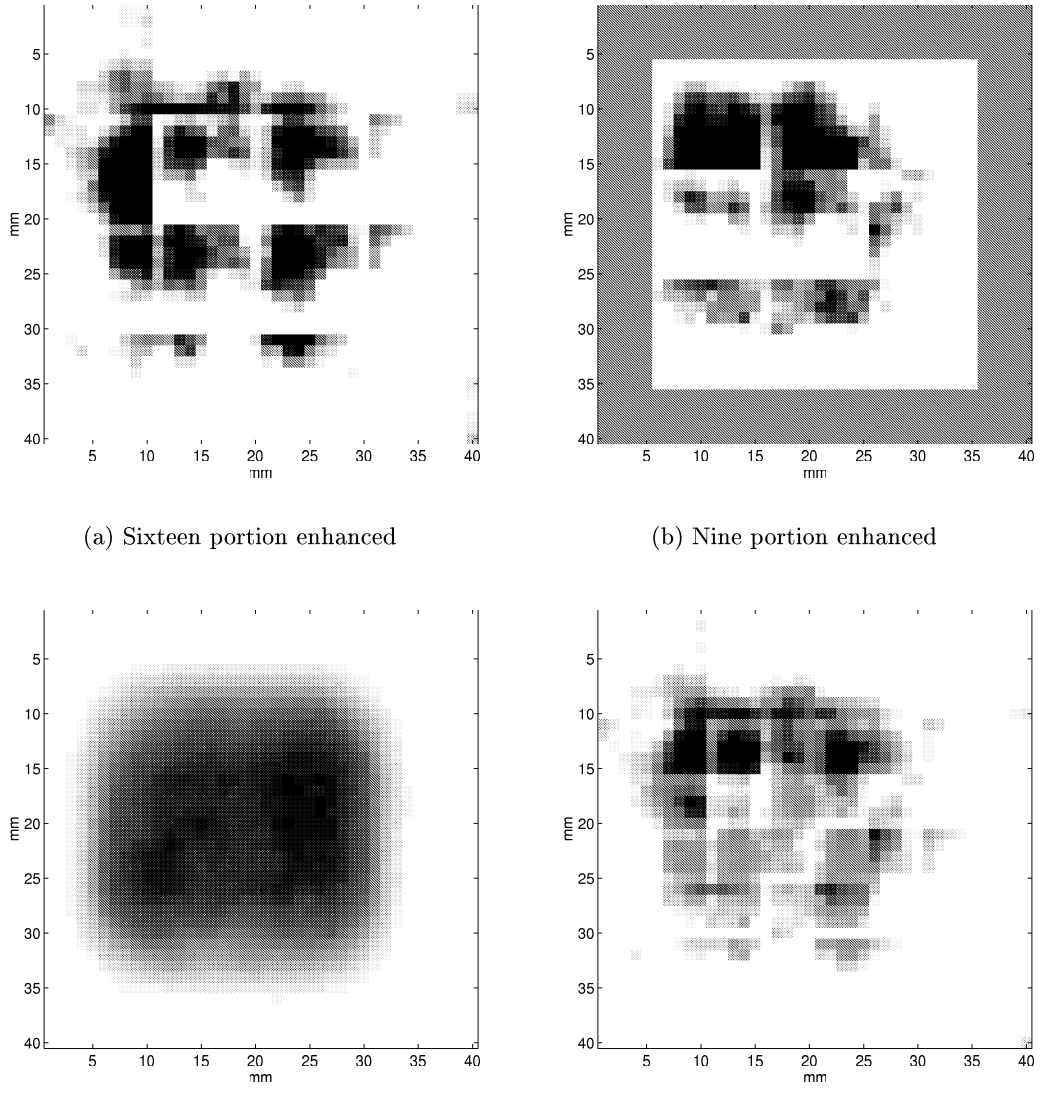
(a) Sixteen portion enhanced

(b) Nine portion enhanced

(c) Original

(d) Combined enhancement

Figure 6.22: 4mm square defect, sixteen and nine portion and combined enhancement.



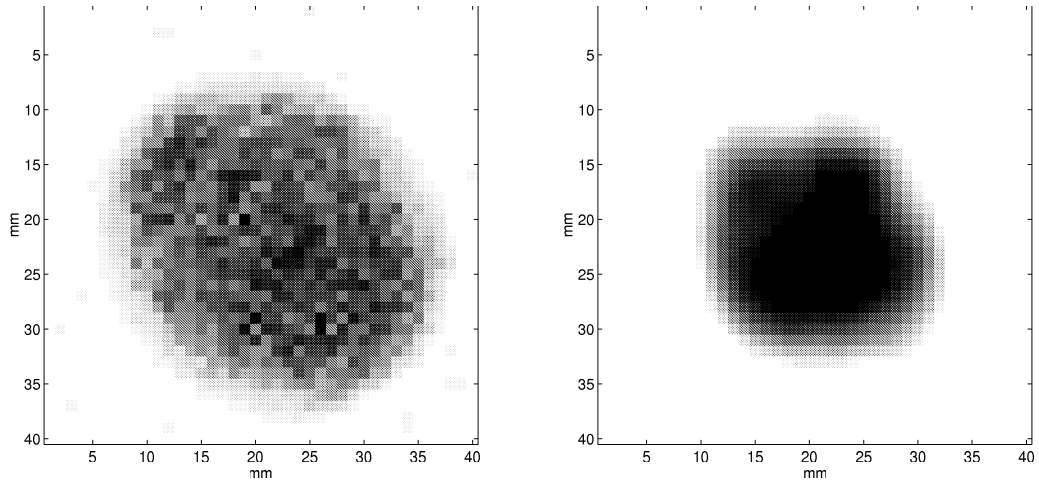
(a) Sixteen portion enhanced

(b) Nine portion enhanced

(c) Original

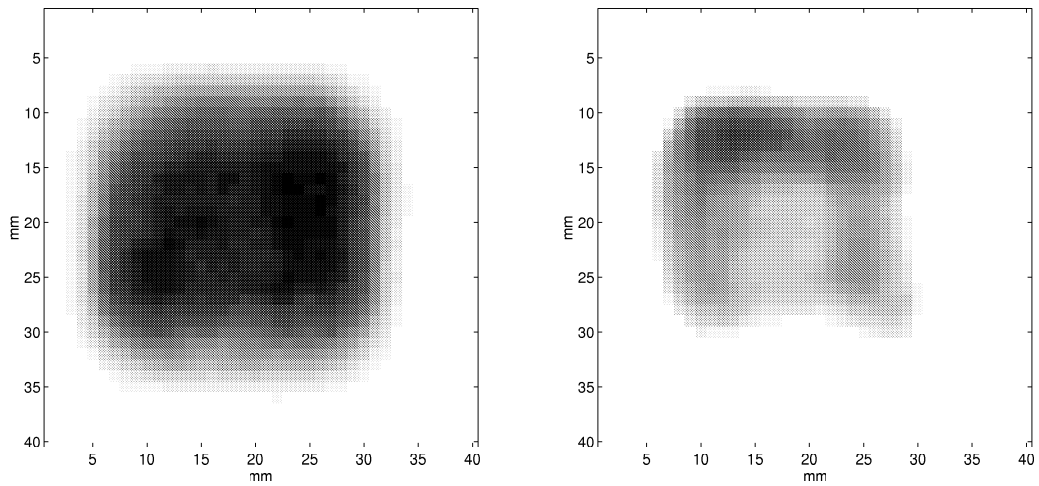
(d) Combined enhancement

Figure 6.23: 20mm square defect, sixteen and nine portion and combined enhancement.



(a) Original (b) Pixel by pixel enhancement

Figure 6.24: Pixel by pixel enhancement for 20mm circular defect.



(a) Original (b) Pixel by pixel enhancement

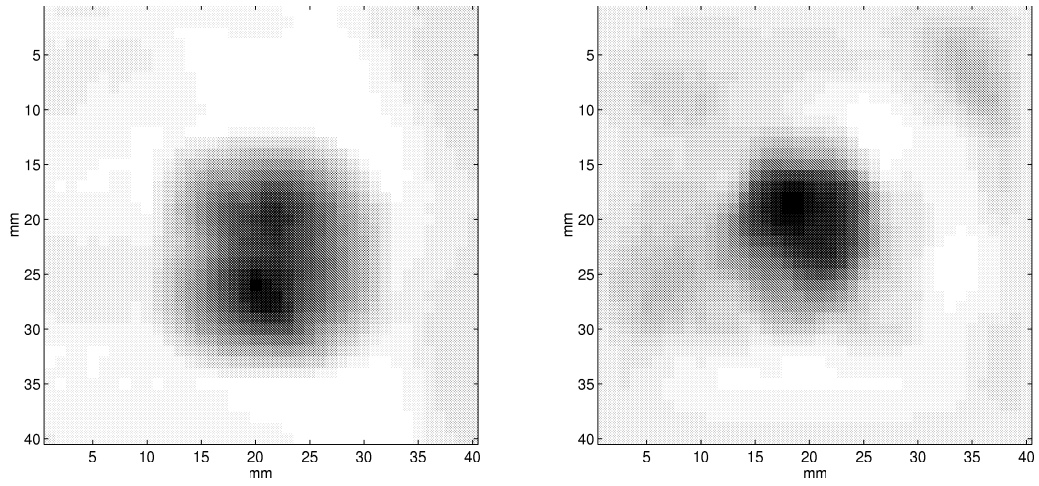
Figure 6.25: Pixel by pixel enhancement for 20mm square defect.

6.6 Validation

The neural network system which performs pixel by pixel image enhancement has been applied to additional validation data. Two carbon fibre reinforced composite samples, both 16ply unidirectional laminate of 2mm thickness, were used. The first sample contained a 12mm square brass defect inclusion 8ply thick located in the centre of the material and introduced during the manufacture of the sample. The second sample similarly contained a 12mm square teflon defect inclusion. Figure 6.26(a) and Figure 6.27(a) illustrate the original C-scan images of the brass and teflon defects.

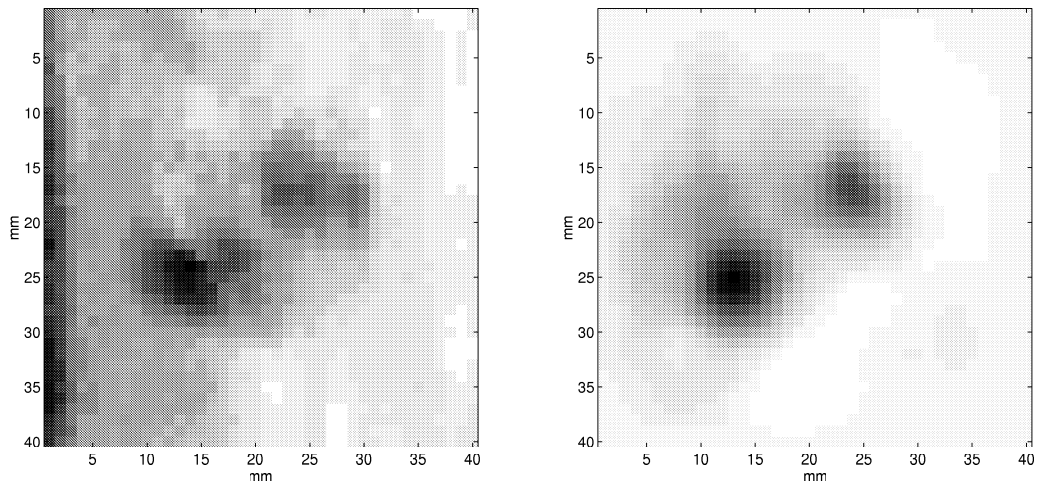
Figure 6.26 demonstrates that the neural network system is able to enhance original C-scan images, in that the image size has been reduced to be closer to the actual dimensions. Note that this was achieved using data taken from different materials than that used to train the neural network, but the processing still allowed a more accurate representation of the actual defect size to be obtained.

Because the neural network has been trained on C-scan images which have been generated by selecting a peak within the ultrasonic waveforms which occurs due to the presence of the defect, the system is only able to enhance C-scan images which have been generated in a similar fashion. Hence only materials with similar ultrasonic properties to that of the material used to train the neural network may produce good quality enhanced images. Thus the original image of the teflon defect, shown in Figure 6.27(a), is a poor quality image as the presence of the defect seemed to disperse the backwall reflection but with only a relatively small additional peak due to the reflection from the defect itself. This caused the image to appear to have two defects. The enhancement shown in Figure 6.27(b) has removed the gradual change in background colour caused by the sample not being exactly level with respect to the transducer and has enhanced the defect into two separate smaller defects.



(a) Original (b) Pixel by pixel enhancement

Figure 6.26: Pixel by pixel enhancement for the 12mm brass defect.



(a) Original (b) Pixel by pixel enhancement

Figure 6.27: Pixel by pixel enhancement for the 12mm teflon defect.

6.7 Summary

In this chapter, a neural network approach to C-scan image enhancement has been developed. The ability of a neural network to determine the correct defect size was demonstrated with line-scan images. This lead to the development of a neural network which took small portions of original C-scan images and generated an enhanced sub-image. By duplicating the actions of this neural network over the entire C-scan image, a neural network system was produced which could enhance the complete C-scan image, thus generating an image which illustrates more accurately the actual defect size. The process thus reduces the effects caused by finite ultrasonic beam sizes, and it is thought that such a technique might have wide application to other forms of defect imaging schemes.

Chapter 7

A Neural Network approach to Voltage Tomography

7.1 Introduction

There are applications where the object to be inspected is located in a hostile environment. In the nuclear sector, for example, metal pipework containing radioactive liquids requires regular monitoring for signs of corrosion. However, such a harsh environment precludes a human from making such inspections. A possible solution to this problem is to permanently attach a sensor device to the pipe and monitor the sensor's response remotely.

A method for the detection of corrosion in metals is to monitor the change in resistivity [116, 117]. This method of detection is called the Field Signature Method (FSM) or Electric Fingerprint Method and was originally patented by the Center of Industrial Research in 1985. The FSM technique employs potential drop or voltage tomography [2, 118–120] whereby a uniform direct current voltage field is generated along the section of pipe under inspection and a sensor device, which contains an array of metal pins, is used to measure the potential drop across various pin pairs. This effectively measures the resistance between the pins and any localised corrosion located beneath certain pin pairs will locally increase the resistance causing the potential drop to change and the corrosion to be detected.

BNFL[1] have been assessing a commercial corrosion detection system which applies the FSM technique for the remote monitoring of metal pipework, specifically for the detection of localised corrosion (referred to as pits or defects). Although visual inspection of the potential drop values can be used to determine the location of localised pits or defects, the depth of such defects cannot be so easily determined. Thus a neural network approach is investigated to determine both the corrosion location and depth.

All of the voltage tomography data detailed in this chapter was supplied by BNFL. The data was generated by two different methods. Initially, simulated data was produced using finite element analysis and has been used as training and testing data for various neural networks. The second method utilizes experimental data, whereby artificial defects simulate localised corrosion, and is to be used as validation data for the trained neural networks.

This chapter is divided in four parts. The first details the experimental work performed by BNFL, and includes a description of the apparatus used, the data collection procedures and the pre-processing technique used. This is followed by a description of the voltage tomography databases, supplied by BNFL, together with the manipulations, performed by the author, upon the databases to produce relevant neural network training, testing and validation data sets. The third part of this chapter details the various neural networks developed for the classification of defect depth and defect location and the neural network which produces images of the pipework. Finally the neural networks previously developed are validated with the experimental data.

[1]British Nuclear Fuels Ltd.

7.2 Experimentation

This section details the experimentation performed by BNFL for the generation of the voltage tomography data. The two different procedures, adopted for data collection, are initially described. This is followed by the specific details of data collection, including a description of the various defect locations and depths used. Finally, the pre-processing technique used for all of the voltage tomography data is explained.

7.2.1 Experimental Apparatus

The instrumentation used by BNFL was manufactured by CorrOcean Ltd. and consisted of two main components, the first of which is the sensor device. The sensor device, constructed as a snap-fit clamp-on system for quick and easy attachment to pipework, consists of an array of sensing pins, a pair of reference pins and two voltage collars. There are 64 sensing pins, arranged in an 8 by 8 square array, with pins being spaced 10mm apart. Every pin is spring loaded and gold plated to allow maximum contact with the pipe under inspection. Although the sensor array contains 64 pins, only the central 16 pins are used for data collection, as illustrated in Figure 7.1. The sensor device is attached to the side of the pipework to be inspected as illustrated by Figure 7.2. Not shown are the pair of reference pins located away from the main array of sensing pins but within the uniform voltage field created by the two voltage collars.

The second component of the instrumentation is the equipment used to measure the various pin potentials. The sensor device is directly connected to the FSM direct current feeder (DCF) which can both supply the direct current voltage field along the pipe via the voltage collars and record and store the potentials at the different pins [121].

The sixteen potentials, measured from the central 16 pins, are used to give twelve potential drop values, calculated by comparing the potentials at pin pairs, 1 and 2, 2 and 3, 3 and 4, 5 and 6, 6 and 7, 7 and 8, 9 and 10, 10 and 11, 11 and 12, 13 and 14, 14 and 15 and finally 15 and 16. In addition a reference pin pair, located away from the sensor array, is used to compensate for changes in the voltage field.

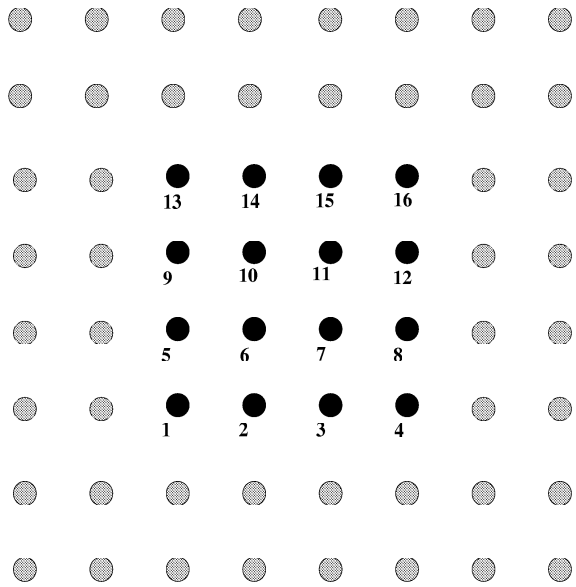


Figure 7.1: The 64 pin sensor array with the 16 central pins highlighted.

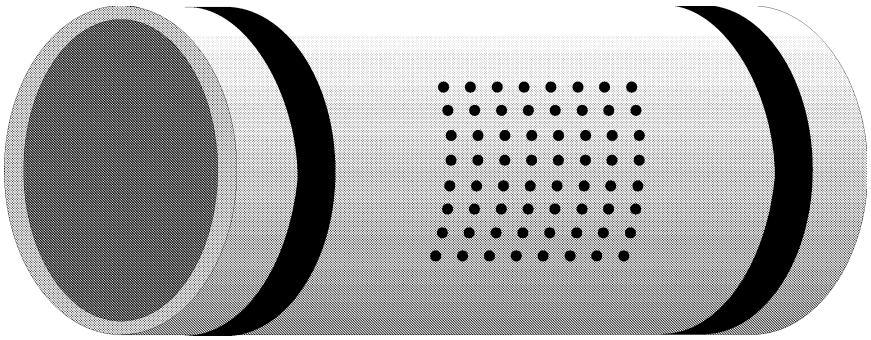


Figure 7.2: Schematic of experimental apparatus.

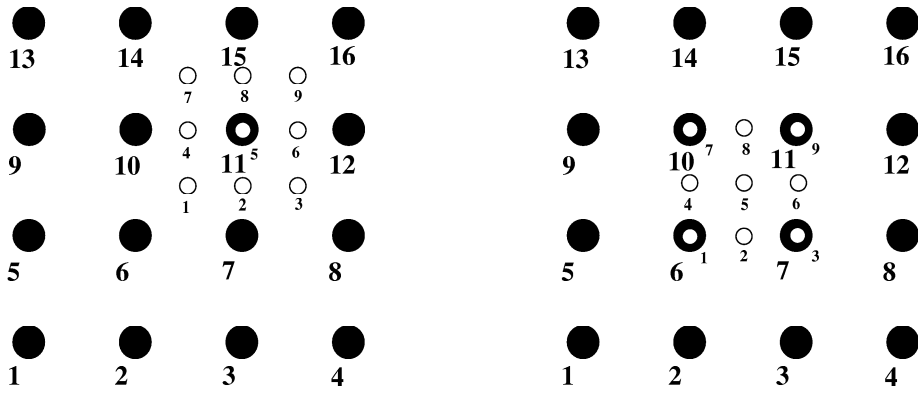
7.2.2 Simulation of the Experimental Apparatus

Initially, the experimental apparatus was simulated using finite element analysis. Ghajarieh *et al* [122] have demonstrated that finite element analysis can be used to evaluate the depth of cracks by modelling potential drop measurements. And so BNFL have generated a set of simulated potential drop data for various defect depths and locations within the sensor array.

7.2.3 Data Collection

A circular defect of diameter 1mm is used for both the simulated and experimental procedure. The relative positioning of the defect locations used, with respect to the central 16 pins, is illustrated in Figure 7.3. As indicated, nine possible locations for the defect exist for both the simulated and experimental procedure. However, comparing the defect positions of Figure 7.3(a) and Figure 7.3(b), highlights a difference in the exact locations used for the two different procedures of data collection.

The simulated defects consists of 19 defect depths, ranging from 0.05cm to 0.95cm (the sample thickness was 1cm), for each of the nine defect locations illustrated in Figure 7.3(a). The experimental defects consists of 17 defect depths, ranging from 0.05cm to 0.85cm, for three of the nine locations shown in Figure 7.3(b). Specifically, the three defect locations used are locations 2, 5 and 6.



(a) Simulated locations (b) Experimental locations

Figure 7.3: Defect locations within the 16 central pins.

7.2.4 Data Pre-processing

As previously described the central 16 pins are used to record a set of sixteen potentials which are then manipulated to calculate twelve potential drop values. Each one of the twelve potential drop or voltage values is translated into fingerprint coefficients which compensates the potential values according to any changes in the reference voltage. The fingerprint coefficient $\mathrm{Fc}_{V_a^t}$ for the voltage at array pin pair a measured at time t is given by

$$\mathrm{Fc}_{V_a^t} = \left(\left(\frac{V_r^{t=0}}{V_a^{t=0}} \times \frac{V_a^t}{V_r^t} \right) - 1 \right) \times 1000 \tag{7.1}$$

where the voltage across the reference pin pair at the start ($t = 0$) is $V_r^{t=0}$, the voltage across the array pin pair at the start is $V_a^{t=0}$, the measured voltage of the reference pin pair at time t is V_r^t and the measured voltage of pin pair a at time t is V_a^t. The $\mathrm{Fc}_{V_a^t}$ is measured in parts per thousand.

7.3 The Voltage Tomography Databases

BNFL supplied two voltage tomography databases, one of simulated data and the other of experimental data. Each database contains a series of twelve fingerprint coefficients for various defect depths and locations. The simulated database consists of nine defect locations, as shown in Figure 7.3(a), and for each location there are 19 defect examples with each example having a different depth ranging from 0.05cm to 0.95cm. In addition there is one example which contains no defect. The experimental database comprises of three of the nine locations shown in Figure 7.3(b), specifically locations 2, 5 and 6. For each location there are 17 defect examples, again each example has a different depth but the range is between 0.05cm and 0.85cm.

Because there is a difference between the defect locations within the simulated database and that of the experimental database (see Figure 7.3), the original simulated database is transformed to give another simulated database which reflects the defect locations of the experimental apparatus. This is possible because the comparison of fingerprint coefficients for different defect locations has demonstrated that the data is symmetrical. Hence, by applying symmetry to the simulated data for defect locations 1, 2, 4 and 5, all nine experimental defect locations can be constructed. This produces another simulated database, but with defect locations which reflect the nine possible experimental locations shown in Figure 7.3(b).

The neural network training and testing data sets are exclusively constructed from the simulated databases. Hence a training data set will be formed by taking one of the simulated databases, re-scaling the inputs and adding output target classifications for each example. The target classifications used ultimately depend on the neural network's task. For the location of defects, as there are only nine unique defect positions, nine target outputs are used to indicate the location of the defect. However, for sizing of the defect (estimating the depth) a single target output is used to encode the actual depth onto the linear range 0 to 1 where 0 represents 'no defect' and 1 represents 1cm (sample thickness).

7.4 Classification of Defect Depth and Location

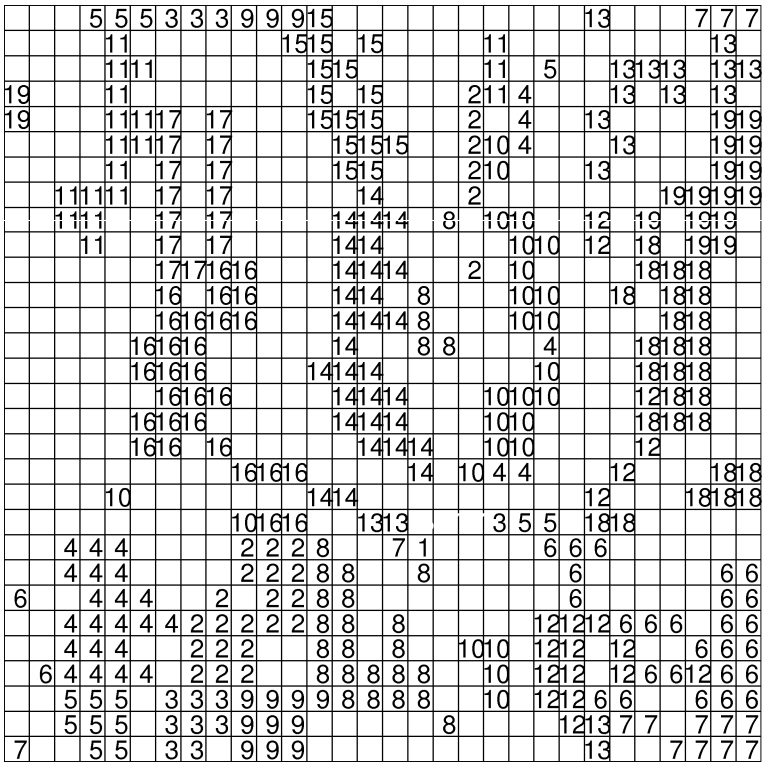
Initially two different neural networks were developed, one to classify defect depth and the other for defect location. This was extended to produce a single neural network which generated an image of the metal pipework under inspection, indicating both the depth and location of localized defects. Note that all of the neural networks presented in this section are trained and tested with the simulated data. Validation of the developed neural networks, with the experimental data, is given in the next section.

7.4.1 Initial Data Analysis

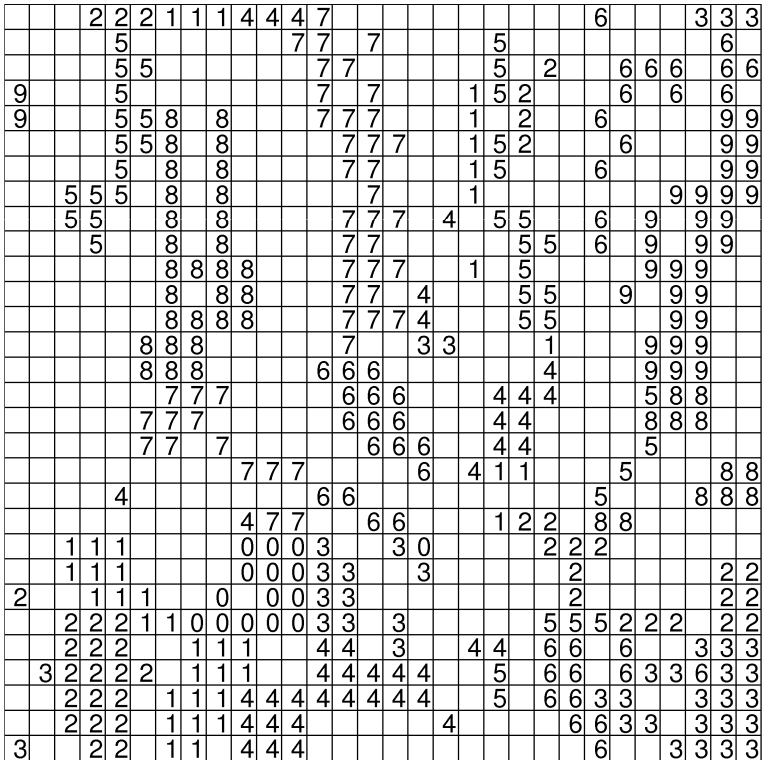
Initial data analysis was performed on the original simulated database and utilized a SOM neural network. Several SOM neural networks were applied and specifically produced maps of 20 by 20 and 30 by 30 outputs, although only the results of the 30 by 30 maps are given here. A square neighbourhood function was chosen and had an initial width, μ_o, of 20 which decreased to 1. The initial learning rate, η_o, was 0.06. The maps were wrapped both horizontally and vertically and the training was stopped after 30 epochs (5,700 cycles).

Figure 7.4 illustrates two 30 by 30 SOM maps. The first map, Figure 7.4(a) has labelled the outputs according to defect depth. The labels range from 1 to 19 indicating progressively deeper defects with a zero label representing a 'no defect'. The second map, Figure 7.4(b), has labelled the outputs according to defect location with a label range of between 1 to 9 representing one of the nine possible locations, again a zero label indicates a 'no defect'.

Both maps illustrate reasonably good clustering of defect depth and location, although the separation between clusters is sometimes less than ideal. However, they do suggest that a BP neural network should be able to classify both depth and location.



(a) Size



(b) Location

Figure 7.4: SOM analysis for defect size and location data.

7.4.2 Practical Issues

Because the simulated database contains only one example of each defect depth at every defect location, then to achieve maximum performance with the experimental validation data the complete simulated database was used to form the training data sets. Testing data sets were created by adding a small amount of random noise to the training data. To balance the simulated database, an additional 18 examples of a 'no defect' pattern were included to give a total of 190 patterns (19 examples of each of the nine defect locations and 19 examples of a 'no defect'). In addition to the fingerprint coefficient pre-processing, previously described, the tomographic data will be re-scaled before being presented to a neural network as either training or testing. The reasons for this re-scaling are detailed in Section 2.7.3.

Note that for neural networks classifying defect depth, only TP and FP classification categories exist. Thus only one performance measure exists for comparison purposes, and that is the positive predictive value (which is effectively a hundredth of the total percentage correct metric).

7.4.3 Neural Network Optimization

As before, the optimum training parameters associated with a BP neural network are determined by an heuristic search for each different classification problem. For optimization the learning rates used varied from 0.001 to 0.3 and the momentum from 0.3 to 0.9. For depth classification, the number of hidden neurons varied from 5 to 15 (applying equation 2.45 gives a range from 6 to 61) together with twelve inputs and one output neuron. For the location classification, the number of hidden neurons varied from 5 to 15 (equation 2.45 gives a range from 24 to 159), however, the number of output neurons used was nine. This gave, for both defect depth and location, 64 different neural networks with various values for momentum, learning rate and numbers of hidden neurons. Also note that the standard BP algorithm was used with periodic updating of weights and the bipolar activation function. The optimum values found were as follows, a learning rate of 0.01, a momentum of 0.9 with 7 hidden neurons for depth and 10 hidden neurons for location. The stop training region was upper bound by 1250 epochs.

7.4.4 Classifying Defect Depth

The optimized neural network for defect depth classification, with a learning rate of 0.01, a momentum of 0.9 and an architecture of 12 input, 7 hidden and 1 output neurons is tested with testing data which had either ±2.5% or ±5% of Gaussian random noise added. The results are given in Table 7.1 and Table 7.2[2]. The neural network is able to correctly classify depth for about 56% of the testing examples. In addition, with ±2.5% noisy testing data, the neural network can correctly classify depth to within one size too big or small of the target size for 77% of the examples, while for the ±5% noisy testing data 73% of the examples are sized to within one size away from the target size.

Inspection of the simulated database indicates a contradiction between no corrosion examples and that of the smallest defect depth (0.05cm). Because there is no difference between the fingerprint coefficients of a no corrosion example and that of a 0.05cm defect, the neural network is unable to correctly detect the smaller defect depths. Thus a neural network, configured as the optimum, has been trained and tested with a data set which has had the smallest defect depths (0.05cm) removed. The results are shown in Table 7.3 and indicate a slight improvement with 57% correctly classified and 77% classified to within one size.

In an attempt to improve the testing results, additional no corrosion examples were added (replacing the number of examples removed for defects of depth 0.05cm). However, this reduced the performance, both for correct classification of depth, 44%, and for one size away, 69%. This was further investigated by the removal of all defect depths below 0.35cm and the addition of more 'no defect' examples, both of which did not improve results.

[2]The output Variation with Target indicates the numerical difference between the target and the output. The value of 0.025 being the maximum numerical distance allowable between the output and target before the output and target represent different physical defect sizes. Thus an output which is a numerical distance of 0.05 away from the corresponding target is one size too large or small

Table 7.1: Testing data with ±2.5% noise added.

Perform.	Output Variation with Target					
Measures	0.025	0.030	0.035	0.040	0.045	0.050
TP	101	114	120	128	132	139
FP	79	66	60	52	48	41
PosPV	0.56	0.63	0.67	0.71	0.73	0.77

Table 7.2: Testing data with ±5% noise added.

Perform.	Output Variation with Target					
Measures	0.025	0.030	0.035	0.040	0.045	0.050
TP	102	110	117	123	128	132
FP	78	70	63	57	52	48
PosPV	0.57	0.61	0.65	0.68	0.71	0.73

Table 7.3: Trained with 0.05cm defects removed, testing data with ±2.5% noise.

Perform.	Output Variation with Target					
Measures	0.025	0.030	0.035	0.040	0.045	0.050
TP	97	105	112	120	130	132
FP	74	66	59	51	41	39
PosPV	0.57	0.61	0.65	0.70	0.76	0.77

7.4.5 Classifying Defect Location

The optimized neural network for defect location classification, with a learning rate of 0.01, a momentum of 0.9 and an architecture of 12 input, 10 hidden and 9 output neurons is tested with testing data which has ±2.5% of random noise added as shown in Table 7.4. The transformed simulated database, with the defect locations which reflect the experimental apparatus arrangement, has also been used to train a neural network. Here the 0.05cm defect depths have been removed. The trained neural network was tested with testing data with ±2.5% of random noise added, the results of which are shown in Table 7.5.

Although for both neural networks the total percentage correct values are very good (95+%), the actual percentage of correctly identified (located) defects is 56% for the original simulated data and 64% for the transformed simulated data.

Table 7.4: Testing of the neural network trained with the original simulated database, for the location of the simulated defects, tested with 2.5% noise added.

Perform.	Threshold								
Measures	-0.800	-0.600	-0.400	-0.200	0.000	0.200	0.400	0.600	0.800
TP	151	121	106	96	90	84	78	73	66
TN	941	1302	1377	1377	1377	1377	1377	1377	1377
FP	436	75	0	0	0	0	0	0	0
FN	11	41	56	66	72	78	84	89	96
Sens	0.93	0.75	0.65	0.59	0.56	0.52	0.48	0.45	0.41
Spec	0.68	0.95	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PosPV	0.26	0.62	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FalAR	0.32	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
% Corr	70.96	92.46	96.36	95.71	95.32	94.93	94.54	94.22	93.76

Table 7.5: Testing of the neural network trained with the transformed simulated database, for the location of the experimental defects, tested with 2.5% noise added.

Perform.	Threshold								
Measures	-0.800	-0.600	-0.400	-0.200	0.000	0.200	0.400	0.600	0.800
TP	133	113	103	99	92	89	84	78	72
TN	1215	1458	1471	1473	1476	1476	1476	1476	1476
FP	261	18	5	3	0	0	0	0	0
FN	11	31	41	45	52	55	60	66	72
Sens	0.92	0.78	0.72	0.69	0.64	0.62	0.58	0.54	0.50
Spec	0.82	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PosPV	0.34	0.86	0.95	0.97	1.00	1.00	1.00	1.00	1.00
FalAR	0.18	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
% Corr	83.21	96.98	97.16	97.04	96.79	96.60	96.30	95.93	95.56

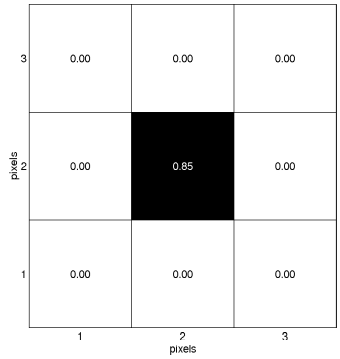
7.4.6 Defect Imaging

For the imaging of defects, the nine target outputs were used to indicate the defect location, however, the targets are no longer binary and the active output encodes the actual depth of the defect as was previously done with the single output of the depth classification neural network.

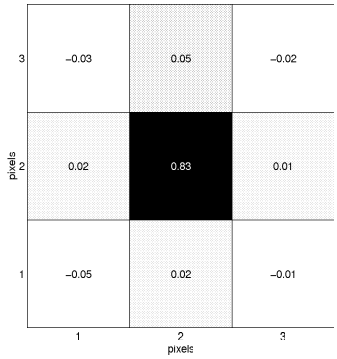
The optimum neural network consisted of 12 inputs, 25 hidden and 9 output neurons, with a learning rate of 0.01 and a momentum of 0.9. The normal simulated database was used for training data, with ±2.5% noise added for testing. Table 7.6 indicates the output of the nine neurons used to form an image, when presented with testing data of various defect depths at location 5. The testing results for all defect locations have shown that using a decision threshold of 0.1, all the defects larger than 0.6cm are correctly located and defects larger than 0.80cm are also correctly sized to within ±0.1cm. Images for a 0.85cm and a 0.95cm deep defect are given in Figure 7.5 along with their target locations.

Table 7.6: The nine image neuron's output for various defect depths at location 5.

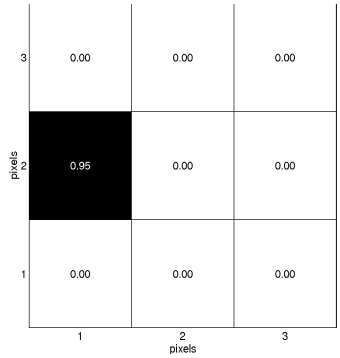
Target	Actual neuron outputs for various defect depths								
Depth	Out 1	Out 2	Out 3	Out 4	Out 5	Out 6	Out 7	Out 8	Out 9
0.05	-0.01	0.38	0.17	-0.05	0.01	0.03	-0.01	0.03	0.02
0.10	0.18	0.01	0.03	-0.01	0.22	-0.02	0.54	0.01	0.01
0.15	-0.02	0.03	0.01	0.22	0.02	0.43	-0.01	0.01	0.04
0.20	0.30	0.02	-0.02	0.01	0.05	-0.03	0.04	0.29	0.02
0.25	0.33	0.01	-0.04	0.05	0.01	0.03	-0.02	0.01	0.03
0.30	-0.04	0.03	-0.01	0.47	0.03	0.01	0.02	0.18	-0.02
0.35	0.01	0.02	0.59	0.05	-0.01	0.03	0.05	-0.04	0.01
0.40	-0.03	0.23	0.01	-0.03	0.01	0.04	-0.02	0.49	0.02
0.45	0.03	0.03	-0.01	0.02	0.14	0.01	-0.04	0.21	0.41
0.50	0.27	-0.01	0.01	0.04	0.21	0.58	0.02	0.03	0.17
0.55	-0.02	0.42	0.02	0.32	0.27	0.03	0.01	-0.02	0.04
0.60	0.46	0.03	0.01	0.02	0.32	0.21	0.02	0.01	0.03
0.65	0.02	-0.03	0.01	0.01	0.45	0.02	0.03	-0.03	-0.01
0.70	0.03	0.01	0.05	0.02	0.62	0.02	0.02	0.01	0.04
0.75	-0.02	0.03	-0.03	-0.02	0.59	0.01	0.02	0.03	0.01
0.80	-0.01	0.04	0.02	0.01	0.71	0.02	0.02	0.02	0.02
0.85	-0.05	0.02	-0.01	0.02	0.83	0.01	-0.03	0.05	-0.02
0.90	0.01	0.03	0.01	-0.03	0.89	-0.01	0.03	0.02	0.01
0.95	0.02	-0.01	0.04	-0.01	0.97	0.01	0.01	0.03	-0.01



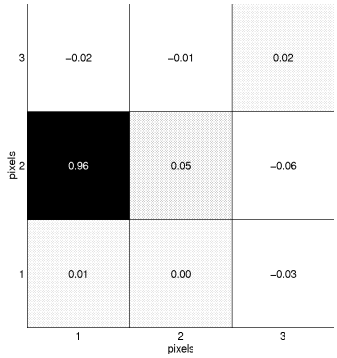
(a) Target for 0.85cm defect



(b) Image for 0.85cm defect



(c) Target for 0.95cm defect



(d) Image for 0.95cm defect

Figure 7.5: Images of two defects illustrating both location and size.

7.5 Validating with Experimental Data

The validation of the three previously developed neural networks, detailed in Section 7.4.4 to Section 7.4.6, has been performed using the experimental database.

Table 7.7 illustrates the results from the neural network for defect depth classification for the various defect depths of the three different experimental locations. In general the depth estimation by the neural network is larger than the target depth. Interestingly, the experimental defect location which the neural network is able to classify the defect depth with the minimum error is location 2, which is not actually represented by the transformed simulated data.

For the classification of location, using the neural network trained with defect locations defined by Figure 7.3(a), the following results were achieved. Examples of experimental location 2 (Figure 7.3(b)) was incorrectly classified by the neural network as location 2 of Figure 7.3(a) for all defect depths above 0.35cm. Experimental location 5 was correctly classified as simulated location 1 for all defect depths above 0.25cm. And the neural network was unable to classify experimental location 6.

However for the neural network trained with the transformed simulated data (transformed to directly relate to the experimental locations). Location 2 was correctly classified as location 2 for defect depths above 0.3cm. Location 5 was incorrectly classified as location 4 for depths above 0.4cm. And location 6 was "unsure" indicating either location 2, 5 or 7 for defect depths between 0.75cm and 0.85cm and gave negative results for smaller depths.

Finally, the combined image generation for the experimental data produced correct location for the defect positioned at location 2 but over estimated the actual size of the defect for sizes larger than 0.3cm. Defect location 5 was incorrectly classified as location 1 or 6, again the size was typically over estimated. And finally defect location 6 produced a "confused" response indicated locations 4, 5 and 6 and occasionally locations 1 and 7.

Table 7.7: Classification of experimental defect depth.

Target Depth	Actual Output		
	Location 2	Location 5	Location 6
0.05	0.19	0.23	0.74
0.10	0.16	0.45	-0.12
0.15	0.13	0.44	-0.12
0.20	0.40	0.70	-0.02
0.25	0.61	0.74	0.94
0.30	0.39	0.77	0.93
0.35	0.40	0.76	0.97
0.40	0.25	0.77	-0.11
0.45	0.47	0.76	0.93
0.50	0.37	0.81	-0.12
0.55	0.60	0.78	-0.12
0.60	0.59	0.81	0.85
0.65	0.74	0.83	0.97
0.70	0.80	0.85	0.94
0.75	0.81	0.85	-0.12
0.80	0.83	0.86	0.82
0.85	0.84	0.87	0.99

7.6 Summary

In this chapter, neural networks have been applied to voltage tomography data. Initially the experimental procedures performed by BNFL to produce the voltage tomography data were illustrated. Neural networks have been trained with simulated data to classify both defect depth and location. This lead to the development of a neural network for the image reconstruction of the pipework under inspection. Finally the various neural networks were validated with experimental data. The system could generate images, which both located and sized a defect correctly for defects which were greater than half the pipe thickness.

Chapter 8

Discussion and Conclusions

8.1 Discussion

This thesis has addressed the application of neural network imaging for several non-destructive testing techniques. In particular, novel neural network systems have been devised for both image reconstruction, applied to ultrasonic and voltage tomography, and image enhancement, used on ultrasonic C-scans. The neural network systems have performed defect detection, location and sizing.

The first application utilized a neural network system for the image reconstruction of advanced composite materials using ultrasound tomography. This involved the design and development of both the sensor array and the instrumentation for the collection of the ultrasonic tomographic data. The sensor array utilized a square geometry and contained a maximum of 40 transducers. The implementation of the instrumentation took two forms. The first required manual operation and permitted sufficient data to be collection for an initial investigation, while the second was automated and allowed a significant increase in the amount of data collected.

A variety of neural network simulations were performed initially, and suggested that the task of defect location and defect sizing was best separated and performed by different neural networks. The location of a given defect size may be performed by a single neural network generating a 4 by 4 pixel output image. For the sizing of the defect, a single neural network with one output gave a classification of defect size and

was shown to be accurate even for validation data. Increasing the image resolution of the neural network performing defect location has been shown to have a problem which affects the training of the neural network. Two methods were developed to overcome this problem and have been investigated. The second method which uses sixteen neural networks for the generation of an output image has allowed the resolution to increase to a 16 by 16 pixel image.

The application of neural networks to conventional ultrasonic NDT has also been investigated. Additional experimental apparatus was used for the collection of C-scan image data. Image enhancement of original C-scan images by a neural network produces images which are more accurate in terms of the defect size. The system utilizes a single neural network which samples a 10 by 10 pixel sub-image taken from a portion of a much larger original C-scan image and generates a corresponding enhanced 10 by 10 pixel sub-image. By simply repeating this process over a complete C-scan image, an enhanced image may be formed. Initially sixteen unique tiled portions of 10 by 10 pixels were used to enhance the 40 by 40 pixel C-scan images. However, improved results were demonstrated when an additional nine overlapping portions are used. This lead to the final system which moves the 10 by 10 pixel portion through the 40 by 40 pixel C-scan image pixel by pixel. Thus the final system can be easily applied to any size of original C-scan image.

The application of neural networks to voltage tomography for localized corrosion detection was also investigated. Simulated data was used to train and test various neural networks, both for the location and sizing of localized defects. Initially separate neural networks were developed for the location and sizing tasks and were validated with experimental data. This lead to a combined neural network which generated an image, indicating both defect location and depth.

8.2 Conclusions

Image generation using a neural network requires a large number of output neurons, which in turn demands a large amount of varied training data. This, together with the fact, that for most applications, the target image will consist of a small localized defect within a much larger area of 'no defect', significantly complicates the process of neural network training. The main contribution of this work was to demonstrate that a neural network can be trained to generate images, in particular, for non-destructive testing applications.

One of the main aspects of this work was to maximize the output resolution achievable with a neural network system. Initially, 4 by 4 pixel images were reconstructed by a neural network, and allowed different types of defects to be located within various materials. Increasing the output resolution of a single neural network to 16 by 16 pixels was found not to be possible with the given training data. However, a number of schemes were devised by the author to allow such an increase in resolution. The most successful of these used a technique of divide and conquer, whereby several (in this case 16) neural networks were used to generate an image (each neural network giving a different part of the complete image). The 16 by 16 pixel images allowed the estimation of both the size and location of defects. However, validaing the sixteen neural networks with different materials and defect types indicated that these neural networks were more sensitive to the material and defect being used (good quality results only possible with materials and defects with similar ultrasonic behaviour to that used to train the neural networks).

The second application, detailed in Chapter 6, demonstrated that a neural network system was able to perform enhancement of C-scan images, primarily compensating for finite ultrasonic beam size, although additional enhancement abilities were experienced. Thus, a number of conventional image enhancement techniques, which are currently used on C-scan images, could potentially be implemented as a single neural network system. Finally, Chapter 7 has demonstrated that neural network image reconstruction can be applied to non-destructive testing techniques other than ultrasound.

8.3 Future Work

The following are suggested areas for future work which naturally follow from the work presented in this thesis.

- The collection of more defect sizes located in 256 positions (or so) within the scanning area will allow several sets of sixteen neural networks to be trained for various defect sizes. By combining the resulting images a system would be formed which could locate and sizes of various sizes by means of a high resolution image.

- Implementation of the totally-integrated instrumentation system. This would consist of porting both the final neural network system and the pre-processing technique to parallel C++. Also necessary is the design and development of a more powerful pulser card and a multiplexer card which is less susceptible to noise.

- The C-scan image enhancement could be implemented as a single program, which would incorporate both the image segmentation and the sub-image neural network enhancement. This would produce a program that would be able to enhance an size of C-scan image with varying degrees of enhancement (depending on the number of sub-images used, from unique tiling to pixel-by-pixel enhancement).

- Extend the experimental data collection for the voltage tomography corrosion data allowing the neural network system to be trained with experimental rather than simulated data.

- Combine the ultrasonic and voltage tomography together into a new hybrid system for NDT applications.

Appendix A

Derivation for the Derivative of the Sigmoid Activation Function

Unipolar Sigmoid Function

Given the unipolar sigmoid activation function,

$$o_k = f(S_k) = \frac{1}{1 + e^{-S_k}}, \tag{A.1}$$

we can write

$$f'(S_k) = \frac{do_k}{dS_k} = \frac{d}{dS_k}\left(\frac{1}{1 + e^{-S_k}}\right). \tag{A.2}$$

If we define

$$u \triangleq 1 + e^{-S_k}. \tag{A.3}$$

Then by using the chain rule we obtain

$$\frac{do_k}{dS_k} = \frac{do_k}{du} \cdot \frac{du}{dS_k} \tag{A.4}$$

and by noting that

$$o_k = \frac{1}{u} \quad \text{which gives} \quad \frac{do_k}{du} = -\frac{1}{u^2} \tag{A.5}$$

and

$$\frac{du}{dS_k} = -e^{-S_k} \tag{A.6}$$

we obtain

$$\begin{aligned} f'(S_k) &= -\frac{1}{u^2}\left(-e^{-S_k}\right) \\ &= -\frac{1}{\left(1+e^{-S_k}\right)^2} \times -e^{-S_k} \\ &= \frac{e^{-S_k}}{\left(1+e^{-S_k}\right)^2}. \end{aligned} \tag{A.7}$$

This can be rewritten as

$$\begin{aligned} f'(S_k) &= \frac{1}{1+e^{-S_k}} \times \frac{1+e^{-S_k}-1}{1+e^{-S_k}} \\ &= \frac{1}{1+e^{-S_k}} \times \left(\frac{1+e^{-S_k}}{1+e^{-S_k}} - \frac{1}{1+e^{-S_k}}\right). \end{aligned} \tag{A.8}$$

Thus from (A.1) we obtain

$$f'(S_k) = o_k\left(1-o_k\right). \tag{A.9}$$

Bipolar Sigmoid Function

Given the bipolar sigmoid activation function

$$o_k = f(S_k) = \frac{2}{1+e^{-S_k}} - 1 \tag{A.10}$$

we have

$$f'(S_k) = \frac{do_k}{dS_k} = \frac{d}{dS_k}\left(\frac{2}{1+e^{-S_k}} - 1\right). \tag{A.11}$$

Again defining equation A.3 we get

$$o_k = \frac{2}{u} - 1 \quad \text{which gives} \quad \frac{do_k}{du} = -\frac{2}{u^2}. \tag{A.12}$$

Using equation A.4 and A.6 we obtain

$$\begin{aligned} f'(S_k) &= -\frac{2}{u^2}\left(-e^{-S_k}\right) \\ &= -\frac{2}{\left(1+e^{-S_k}\right)^2} \times -e^{-S_k} \\ &= \frac{2e^{-S_k}}{\left(1+e^{-S_k}\right)^2} \end{aligned} \tag{A.13}$$

which can be expressed as

$$\begin{aligned} f'(S_k) &= \frac{2+2e^{-S_k}-2}{\left(1+e^{-S_k}\right)^2} \\ &= \frac{-2}{\left(1+e^{-S_k}\right)^2} + \frac{2+2e^{-S_k}}{\left(1+e^{-S_k}\right)^2} \\ &= \frac{-2}{\left(1+e^{-S_k}\right)^2} + \frac{2}{\left(1+e^{-S_k}\right)} \\ &= \frac{1}{2} - \frac{2}{\left(1+e^{-S_k}\right)^2} + \frac{2}{\left(1+e^{-S_k}\right)} - \frac{1}{2} \\ &= \frac{1}{2}\left(1 - \frac{4}{\left(1+e^{-S_k}\right)^2} + \frac{4}{\left(1+e^{-S_k}\right)} - 1\right) \\ &= \frac{1}{2}\left(1 - \left(\frac{2}{1+e^{-S_k}} - 1\right)^2\right). \end{aligned} \tag{A.14}$$

Thus from (A.10) we obtain

$$f'(S_k) = \frac{1}{2}\left(1 - o_k^2\right). \tag{A.15}$$

Appendix B

Program Code for the Instrument Control Software

Version 1: Self-contained Instrumentation

The program code is divided into four program files and four header files:

udas.c	is the main program subroutine.
manrx.c	gives functional control of the receiver card.
mantx.c	allows control of the pulser.
manmux.c	defines the manual operation of the multiplexer.
manrx.h	header file defining the data structures for the receiver.
mantx.h	header file defining the data structures for pulser.
manmux.h	header file defining the data structues for multiplexing.

```
/*******************************************************\
 *
 *       Name: Udas.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: main program routine
 *
\*******************************************************/

#define WIN31
#define BORLC

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <array.h>
#include <abstarry.h>
#include <string.h>
#include <owl.h>
#include <inputdia.h>
#include <filedial.h>
#include <conio.h>

#include "helpwind.h"
#include "auttomo.h"
#include "mantx.h"
#include "manrx.h"
#include "manmux.h"
#include "udas.h"

#define TRUE -1
#define FALSE 0

BOOL OpenManTx, OpenManRx, OpenManMux;

class TMyApp : public TApplication
{
public:
 TMyApp(LPSTR AName,HINSTANCE hInstance,HINSTANCE hPrevInstance,
  LPSTR lpCmdLine, int nCmdShow)
     :TApplication(AName,hInstance,hPrevInstance,
   lpCmdLine,nCmdShow){};
  virtual void InitMainWindow();
};

_CLASSDEF(TMyWindow)
class TMyWindow : public TWindow
{
public:
  char FileName[MAXPATH];

  TMyWindow(PTWindowsObject AParent, LPSTR ATitle);
  ~TMyWindow();
  virtual BOOL CanClose();
  void SetPenSize(int NewSize);
  virtual void CMManMux(RTMessage Msg) = [CM_FIRST+CM_MANMUX];
  virtual void CMManTx(RTMessage Msg) = [CM_FIRST+CM_MANTX];
  virtual void CMManRx(RTMessage Msg) = [CM_FIRST+CM_MANRX];
  virtual void CMManAll(RTMessage Msg) = [CM_FIRST+CM_MANALL];
};

TMyWindow::TMyWindow(PTWindowsObject AParent, LPSTR ATitle)
  : TWindow(AParent, ATitle)
{
  AssignMenu("COMMANDS");
  DisableAutoCreate();
  Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  Attr.X = -4;
  Attr.Y = -4;
  Attr.W = 1032;
  Attr.H = 686;
  OpenManTx = TRUE;
  OpenManRx = TRUE;
  OpenManMux = TRUE;
}

TMyWindow::~TMyWindow()
{}

BOOL TMyWindow::CanClose()
{
  return MessageBox(HWindow, "Do you want to quit?",
    "QUIT", MB_YESNO | MB_ICONQUESTION) == IDYES;
}

void TMyWindow::CMManMux(RTMessage)
{
 if (OpenManMux) {
  PTWindow MMuxWindow;
  OpenManMux = FALSE;
  MMuxWindow = new ManMuxWindow(this);
  MMuxWindow->Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  MMuxWindow->Attr.X = 5;
  MMuxWindow->Attr.Y = 350;
  MMuxWindow->Attr.W = 210;
  MMuxWindow->Attr.H = 290;
  GetApplication()->MakeWindow(MMuxWindow);
 }
}

void TMyWindow::CMManTx(RTMessage)
{
 if (OpenManTx) {
  PTWindow MTxWindow;
  OpenManTx = FALSE;
  MTxWindow = new ManTxWindow(this);
  MTxWindow->Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  MTxWindow->Attr.X = 5;
  MTxWindow->Attr.Y = 70;
  MTxWindow->Attr.W = 210;
  MTxWindow->Attr.H = 250;
  GetApplication()->MakeWindow(MTxWindow);
 }
}

void TMyWindow::CMManRx(RTMessage)
{
 if (OpenManRx) {
  PTWindow MRxWindow;
  OpenManRx = FALSE;
  MRxWindow = new ManRxWindow(this);
  MRxWindow->Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  MRxWindow->Attr.X = 220;
  MRxWindow->Attr.Y = 70;
  MRxWindow->Attr.W = 780;
  MRxWindow->Attr.H = 600;
  GetApplication()->MakeWindow(MRxWindow);
 }
}

void TMyWindow::CMManAll(RTMessage)
{
if (OpenManTx & OpenManMux & OpenManRx) {
  PTWindow MTxWindow;
  OpenManTx = FALSE;
  OpenManMux = FALSE;
  OpenManRx = FALSE;
  MTxWindow = new ManTxWindow(this);
  MTxWindow->Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  MTxWindow->Attr.X = 5;
  MTxWindow->Attr.Y = 70;
  MTxWindow->Attr.W = 210;
  MTxWindow->Attr.H = 250;
  GetApplication()->MakeWindow(MTxWindow);

  PTWindow MMuxWindow;
  MMuxWindow = new ManMuxWindow(this);
  MMuxWindow->Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  MMuxWindow->Attr.X = 5;
  MMuxWindow->Attr.Y = 350;
  MMuxWindow->Attr.W = 210;
  MMuxWindow->Attr.H = 290;
  GetApplication()->MakeWindow(MMuxWindow);

  PTWindow MRxWindow;
  MRxWindow = new ManRxWindow(this);
  MRxWindow->Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  MRxWindow->Attr.X = 220;
  MRxWindow->Attr.Y = 70;
  MRxWindow->Attr.W = 780;
  MRxWindow->Attr.H = 600;
  GetApplication()->MakeWindow(MRxWindow);
 }
}

void TMyApp::InitMainWindow()
{
  MainWindow = new TMyWindow(NULL, Name);
}

int PASCAL WinMain(HINSTANCE hInstance,
     HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow)
{
    TMyApp MyApp("Ultrasonic Data Acquisition System",
    hInstance,hPrevInstance,lpCmdLine,nCmdShow);
  MyApp.Run();
  return MyApp.Status;
}
```

```
/*******************************************************\
 *
 *       Name: ManRx.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: control receiver card
 *
\*******************************************************/

#define WIN31

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <string.h>
#include <owl.h>
#include <owlrc.h>
#include <inputdia.h>
#include <filedial.h>
#include <static.h>
#include <edit.h>
#include <listbox.h>
#include <button.h>
#include <scrollba.h>
#include <radiobut.h>
#include <groupbox.h>
#include "manrx.h"
#include "c:\boards\smis\uturx02.h"
#include "c:\boards\smis\delayx.h"

//////////////////////////////////////////////////////

void delayx(int ms)
{
    union REGS inregs,outregs;
    long unsigned microsec;

    outregs.h.ah = 0x86;
    microsec = (long unsigned)ms*976L;

    outregs.x.cx = (unsigned)(microsec>>16);
    outregs.x.dx = (unsigned)(microsec&0xffff);

    int86(0x15,&outregs,&inregs);  /* Call BIOS interrupt */

}

//////////////////////////////////////////////////////

extern BOOL OpenManRx;
struct boardparam Rxbd;

BOOL Display = FALSE;
BOOL HOLD = FALSE;
BOOL SIGAVG = FALSE;
BOOL NEXT = FALSE;
BOOL READINGS=FALSE;

int NumSigAvg = 64;
int DisSec = 0;
int PntScroll = 0;
int Points = 600;

unsigned long Data[4000];

PTButton AttenBtn;
PTButton FiltBtn;
PTButton G0startBtn;
PTButton G0stopBtn;
PTButton G0lowBtn;
PTButton G0highBtn;
PTButton G1startBtn;
PTButton G1stopBtn;
PTButton G1lowBtn;
PTButton G1highBtn;
PTButton CalstartBtn;
PTButton CalstopBtn;
PTButton DepstartBtn;
PTButton DepstopBtn;
PTButton IftstartBtn;
PTButton IftstopBtn;
PTButton IftthresBtn;
PTButton IfttrigBtn;
PTButton SettingBtn;
PTButton ReadingBtn;
PTButton TrigBtn;
PTButton CancelBtn;
PTButton HoldBtn;
PTButton SumAvgBtn;

union {
unsigned char m[4000];
long l[1000];
} buf;

RECT Area;
RECT update;


 static LOGPEN lpBlack = {PS_SOLID, 1, 1, RGB(0,0,0)};
 HPEN hBlackPen;


 static LOGPEN lpBlue = {PS_SOLID, 1, 1, RGB(0,0,255)},
  lpRed = {PS_SOLID, 1, 1, RGB(255,0,0)},
  lpGreen = {PS_SOLID, 1, 1, RGB(0,255,0)},
  lpMagenta = {PS_SOLID, 1, 1, RGB(255,0,255)},
  lpCyan = {PS_SOLID, 1, 1, RGB(0,255,255)},
  lpYellow = {PS_SOLID, 1, 1, RGB(255,255,0)};

 HPEN hBluePen;
 HPEN hRedPen;
 HPEN hGreenPen;
 HPEN hMagentaPen;
 HPEN hCyanPen;
 HPEN hYellowPen;



//////////////////

ManRxWindow::ManRxWindow(PTWindowsObject AParent) :
   TWindow(AParent, "Manual Control - Receiver")
{
  DisableAutoCreate();
  Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  Attr.X = 220;
  Attr.Y = 70;
  Attr.W = 780;
  Attr.H = 600;

  PW = 250;
  PRF = 60;

  BASEADD = 0x320;

  Area.left = 10;
  Area.right = 610;
  Area.top = 110;
  Area.bottom = 363;

  AttenBtn = new TButton(this, ID_MANRXBUTT1, "Atten",
             640,20,60,30, FALSE);
  FiltBtn = new TButton(this, ID_MANRXBUTT2, "Filter",
             710,20,60,30, FALSE);

  new TStatic(this, -1, "Gate 0", 642,60,160,16,0);
  new TStatic(this, -1, "Gate 1", 712,60,160,16,0);

  G0startBtn = new TButton(this, ID_MANRXBUTT3, "start",
             640,80,60,30, FALSE);
  G0stopBtn = new TButton(this, ID_MANRXBUTT4, "stop",
             640,120,60,30, FALSE);
  G0lowBtn = new TButton(this, ID_MANRXBUTT5, "low",
             640,160,60,30, FALSE);
  G0highBtn = new TButton(this, ID_MANRXBUTT6, "high",
             640,200,60,30, FALSE);

  G1startBtn = new TButton(this, ID_MANRXBUTT7, "start",
             710,80,60,30, FALSE);
  G1stopBtn = new TButton(this, ID_MANRXBUTT8, "stop",
             710,120,60,30, FALSE);
  G1lowBtn = new TButton(this, ID_MANRXBUTT9, "low",
             710,160,60,30, FALSE);
  G1highBtn = new TButton(this, ID_MANRXBUTT10, "high",
             710,200,60,30, FALSE);

  new TStatic(this, -1, "Calib:", 640,235,160,16,0);
  CalstartBtn = new TButton(this, ID_MANRXBUTT11, "start",
             640,260,60,30, FALSE);
  CalstopBtn = new TButton(this, ID_MANRXBUTT12, "stop",
             710,260,60,30, FALSE);

  new TStatic(this, -1, "Depth:", 640,295,160,18,0);
  DepstartBtn = new TButton(this, ID_MANRXBUTT13, "start",
             640,320,60,30, FALSE);
  DepstopBtn = new TButton(this, ID_MANRXBUTT14, "stop",
             710,320,60,30, FALSE);

  new TStatic(this, -1, "IFT:", 640,355,160,16,0);
  IftstartBtn = new TButton(this, ID_MANRXBUTT15, "start",
```

```
              640,380,60,30, FALSE);
  IftstopBtn = new TButton(this, ID_MANRXBUTT16, "stop",
              710,380,60,30, FALSE);
  IftthresBtn = new TButton(this, ID_MANRXBUTT17, "thres",
              640,420,60,30, FALSE);
  IfttrigBtn = new TButton(this, ID_MANRXBUTT18, "trig",
              710,420,60,30, FALSE);


  new TStatic(this, -1, "Gate 0 Gate 1 Calib Depth Iftrig Piop",
              10,10,600,20,0);
  new TStatic(this, -1, "depth peak depth peak depth peak depth
              peak depth",10,32,540,20,0);

  GateEd = new TEdit(this,ID_MANRXGATE,"",8,54,604,30,20,FALSE);

  DisplayBut = new TButton(this, ID_MANRXBUTT19, "DISPLAY ON",
              238,490,153,30, FALSE);
  HoldBtn = new TButton(this, ID_MANRXBUTT20, "HOLD ON",
              238,450,153,30, FALSE);
  SettingBtn = new TButton(this, ID_MANRXBUTT24, "Settings",
              8,530,90,30, FALSE);
  ReadingBtn = new TButton(this, ID_MANRXBUTT25, "Readings",
              8,490,90,30, FALSE);
  TrigBtn = new TButton(this, ID_MANRXBUTT22, "Trigger",
              532,530,80,30, FALSE);
  CancelBtn = new TButton(this, ID_MANRXBUTT23, "Cancel",
              680,530,60,30, FALSE);

  new TStatic(this, -1, "zoom", 291, 400, 70, 16, 0 );
  new TStatic(this, -1, "In", 238, 420, 20, 16, 0 );
  new TStatic(this, -1, "Out", 361, 420, 30, 16, 0 );
  ZoomRange = new TScrollBar(this, ID_MANRXZOOM,
              258,420,100,0, TRUE);
  DisplayRange = new TScrollBar(this, ID_MANRXDISRAN,
              8,366,605,0, TRUE);
  DisplayMode = new TGroupBox(this, ID_MANRXDMODE,
              "Display Mode",400,400,212,120);
  DisLive = new TRadioButton(this, ID_MANRXDLIVE, "Live",
              420,440,60,24, DisplayMode);
  DisSumAvg = new TRadioButton(this, ID_MANRXDSAVG, "Sig Avg",
              420,470,90,24, DisplayMode);
  SumAvgBtn = new TButton(this, ID_MANRXBUTT21, "64",
              520,470,70,30, FALSE);


 hBlackPen = CreatePenIndirect(&lpBlack);
 hBluePen = CreatePenIndirect(&lpBlue);
 hRedPen = CreatePenIndirect(&lpRed);
 hGreenPen = CreatePenIndirect(&lpGreen);
 hMagentaPen = CreatePenIndirect(&lpMagenta);
 hCyanPen = CreatePenIndirect(&lpCyan);
 hYellowPen = CreatePenIndirect(&lpYellow);
}

void ManRxWindow::SetupWindow()
{

 char FileName[130];
 char TextA[50], TextB[50], TextC[50], TextD[50];
 char Text[200];
 int error = 0;

 DefaultSettings(&Rxbd);
 Rxbd.addBASE = BASEADD;
 Rxbd.boardID = 6;
 error = initBoard(&Rxbd);
 if (error) {
   sprintf(TextA,"Initialisation error %x",error);
   error = 0;
 }
 else sprintf(TextA,"Board initialisation okay");

 error = downloadX(&Rxbd, "utua01.bit", "utub01.bit");
 if (error) {
   sprintf(TextB,"Xilinx download error %d", error);
   error = 0;
 }
 else sprintf(TextB,"Xilinx download okay");

 error = downloadDSP(&Rxbd, "utu02.dsp");
 if (error) {
   sprintf(TextC,"DSP download error %d", error);
   error = 0;
 }
 else sprintf(TextC,"DSP download okay");

 error = update_config_iic(&Rxbd);
 if (error) {
   sprintf(TextD,"IIC initialisation error %d", error);
   error = 0;
 }
 else sprintf(TextD,"IIC initialisation okay");

 updateDSP(&Rxbd);
 DefaultDAC(&Rxbd);


 sprintf(Text, "%s\n%s\n%s\n%s", TextA, TextB, TextC, TextD);
 MessageBox(HWindow, Text, "Receiver Board Setup", MB_OK );

 error = 0;

 if (MessageBox(HWindow, "Load initialisation file?",
       "Receiver", MB_YESNO | MB_ICONQUESTION) == IDYES) {
    strcpy(FileName, "*.int");
    if (GetApplication()->ExecDialog(new TFileDialog(this,
                SD_FILEOPEN, FileName))==IDOK){
     FILE *fd;
     struct boardparam b;
     if ( (fd=fopen(FileName, "rb")) != NULL) {
      if ( 1 != fread( &b, sizeof(b), 1, fd) )
       MessageBox(HWindow, "Initialisation file incomplete",
                "FILE ERROR", MB_OK | MB_ICONEXCLAMATION);
       fclose(fd);
       Rxbd = b;
       error = update_config_iic(&Rxbd);
       if (error) MessageBox(HWindow,
              "update_config_iic error","Warning", MB_OK );
       updateDSP(&Rxbd);
     }
     else MessageBox(HWindow,"file not found",
              "Initialisation",MB_OK | MB_ICONEXCLAMATION);
    }
 }

TWindow::SetupWindow();
ZoomRange->SetRange(0, 2);
ZoomRange->PageMagnitude = 1;
DisplayRange->SetRange(0, 600);
DisplayRange->PageMagnitude = 150;
DisplayRange->LineMagnitude = 10;
DisLive->SetCheck(TRUE);
}


void ManRxWindow::MeasureGates()
{
 char ReadData[130];
 char *ptReadData;


// struct boardparam b;

 measure(&Rxbd);


 sprintf(ReadData, "%7.3f %4d  %7.3f %4d  %7.3f %4d
                    %7.3f %4d  %7.3f %4x",
   (float)((Rxbd.M.s.gate0Depth & 0x1fff)*0.025),
   (Rxbd.M.s.gate0Peak & 0xff),
   (float)((Rxbd.M.s.gate1Depth & 0x1fff)*0.025),
   (Rxbd.M.s.gate1Peak & 0xff),
   (float)((Rxbd.M.s.depthDepth & 0x1fff)*0.025),
   (Rxbd.M.s.depthPeak & 0xff),
   (float)((Rxbd.M.s.depthDepthProcessed & 0x1fff)*0.025),
   (Rxbd.M.s.depthPeakProcessed & 0xff),
   (float)((Rxbd.M.s.iftrgDepth & 0x1fff)*0.025),
   (Rxbd.M.s.piopstate & 0xff) );

 ptReadData = ReadData;
 GateEd->SetText(ptReadData);
}

void ManRxWindow::Paint(HDC PaintDC, PAINTSTRUCT _FAR &)
{
int g0start, g0stop, g0low, g0high;
int g1start, g1stop, g1low, g1high;
int i, j, k, count, err, var0, n_pnts;
int ulim, llim;
unsigned char *data;
char chr;
int pixperpnt;

int sloop;
int loop;
int cloop;

unsigned long temp;
int index = 0;


 SelectObject(PaintDC, hBlackPen);
 MoveTo(PaintDC, 8, 108);
 LineTo(PaintDC, 612, 108);
 LineTo(PaintDC, 612, 365);
 LineTo(PaintDC, 8, 365);
 LineTo(PaintDC, 8, 108);
```

```
if (Display) {

       if ( SIGAVG && !HOLD && NEXT) {

        for (loop=0; loop<4000; loop++)
              Data[loop] = 0;

        for (cloop=0; cloop<NumSigAvg; cloop++) {
              enASCAN(&Rxbd);
               readASCAN(&Rxbd, buf.l);
              disASCAN(&Rxbd);
              data = buf.m;
              for (loop=0; loop<4000; loop++)
               Data[loop] += (unsigned long) *(data++);
        }

        for (loop=0; loop<4000; loop++) {
               temp = (unsigned long) (Data[loop] / NumSigAvg);
              Data[loop] = temp;
             }
      }


      if ((DisSec == 0) && !HOLD && !SIGAVG) {

        for (loop=0; loop<4000; loop++)
              Data[loop] = 0;

         enASCAN(&Rxbd);
         readASCAN(&Rxbd, buf.l);
        disASCAN(&Rxbd);
        data = buf.m;
        for (loop=0; loop<4000; loop++) {
          Data[loop] = (unsigned long) *(data++);
        }
      }

      update.left = 10 + (50 * DisSec);
      update.right = 60 + (50 * DisSec);
      update.top = 110;
      update.bottom = 363;

      DisSec++;
      if (DisSec > 11) DisSec = 0;

      pixperpnt = (int) (Points/600);
      count = Points-2;

       g0start = (((Rxbd.S.s.gate0_start>>3) & 0x1fff)/pixperpnt)
                  - (int)(PntScroll/pixperpnt);
       g0stop = (((Rxbd.S.s.gate0_stop>>3) & 0x1fff)/pixperpnt)
                  - (int)(PntScroll/pixperpnt);

       g0low = (int) (0.5*(255
                  - ((Rxbd.S.s.gate0_low_thresh>>8) & 0xff)));
       g0high = (int) (0.5*(255
                  - ((Rxbd.S.s.gate0_high_thresh>>8) & 0xff)));

       SelectObject(PaintDC, hMagentaPen);
       MoveTo(PaintDC, (10+g0start), (110+g0low) );
       LineTo(PaintDC, (10+g0stop), (110+g0low) );
       MoveTo(PaintDC, (10+g0stop), (111+g0low) );
       LineTo(PaintDC, (10+g0start), (111+g0low) );

       MoveTo(PaintDC, (10+g0start), (110+g0high) );
       LineTo(PaintDC, (10+g0stop), (110+g0high) );
       MoveTo(PaintDC, (10+g0stop), (111+g0high) );
       LineTo(PaintDC, (10+g0start), (111+g0high) );


       g1start = (((Rxbd.S.s.gate1_start>>3) & 0x1fff)/pixperpnt)
          - (int)(PntScroll/pixperpnt);
       g1stop = (((Rxbd.S.s.gate1_stop>>3) & 0x1fff)/pixperpnt)
          - (int)(PntScroll/pixperpnt);

       g1low = (int) (0.5*(255
          - ((Rxbd.S.s.gate1_low_thresh>>8) & 0xff)));
       g1high = (int) (0.5*(255
          - ((Rxbd.S.s.gate1_high_thresh>>8) & 0xff)));

       SelectObject(PaintDC, hCyanPen);
       MoveTo(PaintDC, (10+g1start), (110+g1low) );
       LineTo(PaintDC, (10+g1stop), (110+g1low) );
       MoveTo(PaintDC, (10+g1stop), (111+g1low) );
       LineTo(PaintDC, (10+g1start), (111+g1low) );

       MoveTo(PaintDC, (10+g1start), (110+g1high) );
       LineTo(PaintDC, (10+g1stop), (110+g1high) );
       MoveTo(PaintDC, (10+g1stop), (111+g1high) );
       LineTo(PaintDC, (10+g1start), (111+g1high) );

       SelectObject(PaintDC, hRedPen);

       i=0; k=0;
       index=0;

       index+=PntScroll;

       MoveTo( PaintDC, 10, (110+(255 -(int)Data[index])));

       if (pixperpnt > 1) {
          k=pixperpnt;
          do {
             index++;
             ulim = (255- (int) Data[index]);
             llim = (255- (int) Data[index]);
             for (j=1;j<pixperpnt;j++) {
                if ( (255- (int) Data[index]) > ulim)
                    ulim = (255- (int) Data[index]);
                if ( (255- (int) Data[index]) < llim)
                    llim = (255- (int) Data[index]);
                index++;
             }
             LineTo(PaintDC, 10+i, 110 + llim );
             LineTo(PaintDC, (10 + i++), 110 + ulim );
             k += pixperpnt;
          } while (k < count);  //count
        }
       else {
        do {
          LineTo(PaintDC,(10+i++),
               (110+(255-(int)Data[index++])));
        } while ( ++k < count );
        }
      if (!HOLD && !NEXT)
            InvalidateRect(HWindow, &update, TRUE);
 }
}


void ManRxWindow::DefaultSettings(struct boardparam *b)
{
  b->S.s.flags = 0;
  b->S.s.DAC1 = 128<<8;
  b->S.s.XIL0_trig = (1<<14)+(1<<12)+(1<<10);
  b->S.s.XIL1_trig = b->S.s.XIL0_trig;
  b->S.s.calib_time_start = (160*40)<<3;
  b->S.s.calib_time_end = (170*40)<<3;
  b->S.s.gate0_stop = (90*40)<<3;
  b->S.s.gate0_start = (40*40)<<3;
  b->S.s.DAC0 = 240;
  b->S.s.gate0_high_thresh = b->S.s.DAC0<<8;
  b->S.s.gate0_low_thresh = 10<<8;
  b->S.s.iftrg_dest_trigger_line = 1<<13;
  b->S.s.iftrg_stop = (40*40)<<3;
  b->S.s.iftrg_start = (10*40)<<3;
  b->S.s.iftrg_thresh = 128<<8;
  b->S.s.depth_stop = (40*40)<<3;
  b->S.s.depth_start = (30*40)<<3;
  b->S.s.dac_stop = (190*40)<<3;
  b->S.s.gate1_stop = (30*40)<<3;
  b->S.s.gate1_start = (20*40)<<3;
  b->S.s.DAC1 = 240;
  b->S.s.gate1_high_thresh = b->S.s.DAC1<<8;
  b->S.s.gate1_low_thresh = 100<<8;

  b->S.s.flagGate0Thresh=(b->S.s.gate0_low_thresh>>8) & 0xff;
  b->S.s.flagGate1Thresh=(b->S.s.gate1_low_thresh>>8) & 0xff;
  b->S.s.flagIftrgThresh=((b->S.s.iftrg_stop>>3)+3) & 0x1fff;

  b->bit = FALSE;
  b->sng = TRUE;
  b->blank = FALSE;
  b->board = 0;
  b->atten = 15;
  b->filter = 1;
}


void ManRxWindow::HandleButton1Msg(RTMessage)
{
 if (!Display) {
  char InputText[10];
  int err = 0;
  sprintf(InputText, "%4d", Rxbd.atten);
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Attenuation","Enter new Attenuation (in dBs)",
        InputText, sizeof InputText)) == IDOK )
  {
  Rxbd.atten = atoi(InputText);
  err = update_config_iic(&Rxbd);
  if (err) MessageBox(HWindow, "update_config_iic error",
           "Warning", MB_OK );
  }
 }
}
```

```
void ManRxWindow::HandleButton2Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
  int err = 0;

  sprintf(InputText, "%1d", Rxbd.filter);
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
    "Filter","Filter 0 for 2.5MHz, 1 for 5MHz, 2 for other.",
    InputText, sizeof InputText)) == IDOK )
  {
  Rxbd.filter = atoi(InputText);
  err = update_config_iic(&Rxbd);
  if (err) MessageBox(HWindow, "update_config_iic error",
           "Warning", MB_OK );
  }
 }
}

void ManRxWindow::HandleButton3Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.gate0_start>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 0","Start after trigger (us)", InputText,
        sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.gate0_start=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton4Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.gate0_stop>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 0","Stop after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.gate0_stop=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton5Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
  int Var;

  sprintf(InputText, "%3d",
       ((Rxbd.S.s.gate0_low_thresh>>8) & 0xff));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 0","Low threshold (% of screen height)",
       InputText, sizeof InputText)) == IDOK )
  {
  Var = atoi(InputText);
  Var = 1 + (int)(2.55 * (float)Var);
  Rxbd.S.s.gate0_low_thresh = Var<<8;
  Rxbd.S.s.flagGate0Thresh =
        (Rxbd.S.s.gate0_low_thresh>>8) & 0xff;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton6Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
  int Var;

  sprintf(InputText, "%3d",
       ((Rxbd.S.s.gate0_high_thresh>>8) & 0xff));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 0","High threshold (% of screen height)",
       InputText, sizeof InputText)) == IDOK )
  {
  Var = atoi(InputText);
  Var = 1 + (int)(2.55 * (float)Var);
  Rxbd.S.s.DAC0 = Var;
  Rxbd.S.s.gate0_high_thresh = Rxbd.S.s.DAC0<<8;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton7Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.gate1_start>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 1","Start after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.gate1_start=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton8Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.gate1_stop>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 1","Stop after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.gate1_stop=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton9Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
  int Var;

  sprintf(InputText, "%3d",
       ((Rxbd.S.s.gate1_low_thresh>>8) & 0xff));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 1","Low threshold (% of screen height)",
       InputText, sizeof InputText)) == IDOK )
  {
  Var = atoi(InputText);
  Var = 1 + (int)(2.55 * (float)Var);
  Rxbd.S.s.gate1_low_thresh = Var<<8;
  Rxbd.S.s.flagGate1Thresh =
       (Rxbd.S.s.gate1_low_thresh>>8) & 0xff;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton10Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
  int Var;

  sprintf(InputText, "%3d",
       ((Rxbd.S.s.gate1_high_thresh>>8) & 0xff));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Gate 1","High threshold (% of screen height)",
       InputText, sizeof InputText)) == IDOK )
  {
  Var = atoi(InputText);
  Var = 1 + (int)(2.55 * (float)Var);
  Rxbd.S.s.DAC1 = Var;
  Rxbd.S.s.gate1_high_thresh = Rxbd.S.s.DAC1<<8;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton11Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
```

```
  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.calib_time_start>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Calib","Start after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.calib_time_start =
       (int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton12Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.calib_time_end>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Calib","Stop after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.calib_time_end =
       (int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton13Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.depth_start>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Depth","Start after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.depth_start=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton14Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.depth_stop>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Depth","Stop after trigger (us)", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.depth_stop=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton15Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.iftrg_start>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "IFT","Interface trig start after trigger (us)",
       InputText, sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.iftrg_start=(int)((atof(InputText))*40.0)<<3;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton16Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%7.3f",
       (((Rxbd.S.s.iftrg_stop>>3) & 0x1fff)/40.0));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "IFT","Interface trig stop after trigger (us)",
       InputText, sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.iftrg_stop=(int)((atof(InputText))*40.0)<<3;
  Rxbd.S.s.flagIftrgThresh =
       ((Rxbd.S.s.iftrg_stop>>3)+3) & 0x1fff;
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton17Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%4d",
       ((Rxbd.S.s.iftrg_thresh>>8) & 0xff));
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "IFT","Trigger threshold 0..255", InputText,
       sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.iftrg_thresh = ((atoi(InputText))<<8);
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton18Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];
  int trig;
  int newtrig;

  if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
          & 0xf) == 1) trig = 0;
  else if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
          & 0xf) == 2) trig = 1;
  else if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
          & 0xf) == 4) trig = 2;
  else if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
          & 0xf) == 8) trig = 3;
  else trig = -1;

  sprintf(InputText, "%d", trig );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
   "IFT","Destination for trigger; 0, 1, 2, 3 or -1 for none",
    InputText, sizeof InputText)) == IDOK )
  {
  Rxbd.S.s.iftrg_dest_trigger_line = 0;
  newtrig = atoi(InputText);
  if (newtrig>=0)
      Rxbd.S.s.iftrg_dest_trigger_line = 1 << (newtrig+12);
  updateDSP(&Rxbd);
  }
 }
}

void ManRxWindow::HandleButton19Msg(RTMessage)
{
if (Display) {
   Display = FALSE;
   SetWindowText( DisplayBut->HWindow, "DISPLAY ON");
    EnableWindow(AttenBtn->HWindow, TRUE);
    EnableWindow(FiltBtn->HWindow, TRUE);
    EnableWindow(GOstartBtn->HWindow, TRUE);
    EnableWindow(GOstopBtn->HWindow, TRUE);
    EnableWindow(GOlowBtn->HWindow, TRUE);
    EnableWindow(GOhighBtn->HWindow, TRUE);
    EnableWindow(G1startBtn->HWindow, TRUE);
    EnableWindow(G1stopBtn->HWindow, TRUE);
    EnableWindow(G1lowBtn->HWindow, TRUE);
    EnableWindow(G1highBtn->HWindow, TRUE);
    EnableWindow(CalstartBtn->HWindow, TRUE);
    EnableWindow(CalstopBtn->HWindow, TRUE);
    EnableWindow(DepstartBtn->HWindow, TRUE);
    EnableWindow(DepstopBtn->HWindow, TRUE);
    EnableWindow(IftstartBtn->HWindow, TRUE);
    EnableWindow(IftstopBtn->HWindow, TRUE);
    EnableWindow(IftthresBtn->HWindow, TRUE);
    EnableWindow(IfttrigBtn->HWindow, TRUE);
    EnableWindow(SettingBtn->HWindow, TRUE);
    EnableWindow(ReadingBtn->HWindow, TRUE);
    EnableWindow(TrigBtn->HWindow, TRUE);
    EnableWindow(CancelBtn->HWindow, TRUE);
    EnableWindow(SumAvgBtn->HWindow, TRUE);
   }
 else {
    Display = TRUE;
    SetWindowText( DisplayBut->HWindow, "DISPLAY OFF");
```

```
    EnableWindow(AttenBtn->HWindow, FALSE);
    EnableWindow(FiltBtn->HWindow, FALSE);
    EnableWindow(G0startBtn->HWindow, FALSE);
    EnableWindow(G0stopBtn->HWindow, FALSE);
    EnableWindow(G0lowBtn->HWindow, FALSE);
    EnableWindow(G0highBtn->HWindow, FALSE);
    EnableWindow(G1startBtn->HWindow, FALSE);
    EnableWindow(G1stopBtn->HWindow, FALSE);
    EnableWindow(G1lowBtn->HWindow, FALSE);
    EnableWindow(G1highBtn->HWindow, FALSE);
    EnableWindow(CalstartBtn->HWindow, FALSE);
    EnableWindow(CalstopBtn->HWindow, FALSE);
    EnableWindow(DepstartBtn->HWindow, FALSE);
    EnableWindow(DepstopBtn->HWindow, FALSE);
    EnableWindow(IftstartBtn->HWindow, FALSE);
    EnableWindow(IftstopBtn->HWindow, FALSE);
    EnableWindow(IftthresBtn->HWindow, FALSE);
    EnableWindow(IfttrigBtn->HWindow, FALSE);
    EnableWindow(SettingBtn->HWindow, FALSE);
    EnableWindow(ReadingBtn->HWindow, FALSE);
    EnableWindow(TrigBtn->HWindow, FALSE);
    EnableWindow(CancelBtn->HWindow, FALSE);
    EnableWindow(SumAvgBtn->HWindow, FALSE);
    }
 InvalidateRect(HWindow, &Area, TRUE);
}


void ManRxWindow::HandleButton20Msg(RTMessage)
{
if (Display) {
 if (SIGAVG) {
    HOLD = FALSE;
    InvalidateRect(HWindow, &Area, TRUE);
  }

 if (!SIGAVG) {
  if (!HOLD) {
    HOLD = TRUE;
    SetWindowText( HoldBtn->HWindow, "HOLD OFF");
    }
   else {
    HOLD = FALSE;
    SetWindowText( HoldBtn->HWindow, "HOLD ON");
    }
  InvalidateRect(HWindow, &Area, TRUE);
  }
 }
}

void ManRxWindow::HandleButton21Msg(RTMessage)
{
 if (!Display) {

  char InputText[10];

  sprintf(InputText, "%s", "64" );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
     "Signal Averaging","Enter number of waveforms to average",
     InputText, sizeof InputText)) == IDOK )
  {
   SetWindowText(SumAvgBtn->HWindow, InputText);
   NumSigAvg = atoi(InputText);
  }
 }
}

void ManRxWindow::HandleButton22Msg(RTMessage)
{
 if (!Display) {

 char InputText[10];
 int trig;

  trig = 3 & (Rxbd.S.s.XIL0_trig>>(4+0));
  sprintf(InputText, "%i", trig );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Trigger","trigger source for gate0; 0,1,2,3",
       InputText, sizeof InputText)) == IDOK )
    {
     Rxbd.S.s.XIL0_trig = atoi(InputText);
    }

  trig = 3 & (Rxbd.S.s.XIL0_trig>>(4+2));
  sprintf(InputText, "%i", trig );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Trigger","trigger source for gate1; 0,1,2,3",
       InputText, sizeof InputText)) == IDOK )
    {
     Rxbd.S.s.XIL0_trig |= (atoi(InputText))<<2;
    }

  trig = 3 & (Rxbd.S.s.XIL0_trig>>(4+4));
  sprintf(InputText, "%i", trig );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Trigger","trigger source for iftrg; 0,1,2,3",
       InputText, sizeof InputText)) == IDOK )
    {
     Rxbd.S.s.XIL0_trig |= (atoi(InputText))<<4;
    }

  trig = 3 & (Rxbd.S.s.XIL0_trig>>(4+8));
  sprintf(InputText, "%i", trig );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Trigger","trigger source for dac; 0,1,2,3",
       InputText, sizeof InputText)) == IDOK )
    {
     Rxbd.S.s.XIL0_trig |= (atoi(InputText))<<8;
    }

  trig = 3 & (Rxbd.S.s.XIL0_trig>>(4+6));
  sprintf(InputText, "%i", trig );
  if ( GetApplication()->ExecDialog(new TInputDialog(this,
       "Trigger","trigger source for depth; 0,1,2,3",
       InputText, sizeof InputText)) == IDOK )
    {
     Rxbd.S.s.XIL0_trig |= (atoi(InputText))<<6;
    }

  Rxbd.S.s.XIL0_trig |= (atoi(InputText))<<10;
  Rxbd.S.s.XIL0_trig <<=4;
  Rxbd.S.s.XIL1_trig = Rxbd.S.s.XIL0_trig;
  updateDSP(&Rxbd);
 }
}

void ManRxWindow::HandleButton23Msg(RTMessage)
{
 if (!Display) {

  char FileName[130];

  if (MessageBox(HWindow, "Save initialisation file?",
      "Receiver", MB_YESNO | MB_ICONQUESTION) == IDYES) {
    strcpy(FileName, "setup.int");
    if (GetApplication()->ExecDialog(new TFileDialog(this,
      SD_FILESAVE, FileName))==IDOK)
    {
     FILE *fd;
     struct boardparam b = Rxbd;
     if ( (fd=fopen(FileName, "wb")) != NULL) {
       if ( 1 != fwrite( &b, sizeof(b), 1, fd) )
           MessageBox(HWindow, "Save failed", "FILE ERROR",
               MB_OK | MB_ICONEXCLAMATION);
        fclose(fd);
     }
     else MessageBox(HWindow, "Cannot write to file",
              "FILE ERROR", MB_OK | MB_ICONEXCLAMATION);
    }
 }
 CloseWindow();
 OpenManRx = TRUE;
 }
}

void ManRxWindow::HandleButton24Msg(RTMessage)
{
if (!Display) {

char TextA[100];
char TextB[200];
char TextC[200];
char TextD[200];
char TextE[100];
char TextF[100];

char Text[1000];

int trig;
float filt;

 if (Rxbd.filter == 0) filt = 2.5;
  else if (Rxbd.filter == 1) filt = 5.0;
   else filt = 0.0;

sprintf(TextA, "Filter %4.1f MHz\t\tAttenuation %d dBs\n\n",
        filt, Rxbd.atten);
sprintf(TextB, "Gate0: start %7.3f us, stop %7.3f us\nGate0:
        low thresh %3d, upper thresh %3d, trig source %d\n\n",
       ((Rxbd.S.s.gate0_start>>3) & 0x1fff)/40.0,
       ((Rxbd.S.s.gate0_stop>>3) & 0x1fff)/40.0,
       (Rxbd.S.s.gate0_low_thresh>>8) & 0xff,
       (Rxbd.S.s.gate0_high_thresh>>8) & 0xff,
       3 & (Rxbd.S.s.XIL0_trig>>(4+0))
       );

sprintf(TextC, "Gate1: start %7.3f us, stop %7.3f us\nGate1:
       low thresh %3d, upper thresh %3d, trig source %d\n\n",
```

```
       ((Rxbd.S.s.gate1_start>>3) & 0x1fff)/40.0,
       ((Rxbd.S.s.gate1_stop>>3) & 0x1fff)/40.0,
       (Rxbd.S.s.gate1_low_thresh>>8) & 0xff,
       (Rxbd.S.s.gate1_high_thresh>>8) & 0xff,
       3 & (Rxbd.S.s.XIL0_trig>>(4+2))
       );

 if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
        & 0xf) == 1) trig = 0;
  else if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
        & 0xf) == 2) trig = 1;
  else if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
        & 0xf) == 4) trig = 2;
  else if ( ((Rxbd.S.s.iftrg_dest_trigger_line>>12)
        & 0xf) == 8) trig = 3;
  else trig = -1;


sprintf(TextD, "Iftrg: start %7.3f us, stop %7.3f us\nIftrg:
       thresh %3d, dest trig %d, trig source %d\n\n",
       ((Rxbd.S.s.iftrg_start>>3) & 0x1fff)/40.0,
       ((Rxbd.S.s.iftrg_stop>>3) & 0x1fff)/40.0,
       (Rxbd.S.s.iftrg_thresh>>8) & 0xff,
       trig,
       3 & (Rxbd.S.s.XIL0_trig>>(4+4))
       );

sprintf(TextE, "Depth: start %7.3f us, stop %7.3f us,
       trig source %d\n\n",
       ((Rxbd.S.s.depth_start>>3) & 0x1fff)/40.0,
       ((Rxbd.S.s.depth_stop>>3) & 0x1fff)/40.0,
       3 & (Rxbd.S.s.XIL0_trig>>(4+6))
        );

sprintf(TextF, "Calib: start %7.3fus, stop %7.3f us\n",
       ((Rxbd.S.s.calib_time_start>>3) & 0x1fff)/40.0,
       ((Rxbd.S.s.calib_time_end>>3) & 0x1fff)/40.0
        );

sprintf(Text, "%s%s%s%s%s", TextA, TextB, TextC,
       TextD, TextE, TextF);
MessageBox(HWindow, Text, "Receiver Board Settings", MB_OK );
 }
}

void ManRxWindow::HandleButton25Msg(RTMessage)
{
 if (!Display) {
  if (!READINGS) READINGS=TRUE;
    else READINGS=FALSE;
  MeasureGates();
 }
}

void ManRxWindow::CalImageArea()
{

Points = 600+ (600*(ZoomRange->GetPosition()));
if (Points>1800) Points = 1800;
if (Points<600) Points = 600;
if ( (ZoomRange->GetPosition()) == 0 ) {
   PntScroll = 2*(DisplayRange->GetPosition());
  }
if ( (ZoomRange->GetPosition()) == 1 ) {
   PntScroll = (int) (1*(DisplayRange->GetPosition()));
  }
if ( (ZoomRange->GetPosition()) == 2 ) PntScroll = 0;

InvalidateRect(HWindow, &Area, TRUE);
}

void ManRxWindow::HandleZoomBarMsg(RTMessage)
{
HOLD = TRUE;
if (!NEXT) SetWindowText( HoldBtn->HWindow, "HOLD OFF");
CalImageArea();
}

void ManRxWindow::HandleDisRanBarMsg(RTMessage)
{
HOLD = TRUE;
if (!NEXT) SetWindowText( HoldBtn->HWindow, "HOLD OFF");
CalImageArea();
}

void ManRxWindow::HandleDisModeMsg(RTMessage)
{
 if ( DisLive->GetCheck() == BF_CHECKED ) {
   EnableWindow( SumAvgBtn->HWindow, FALSE);
   SetWindowText( HoldBtn->HWindow, "HOLD ON");
   HOLD = FALSE;
   NEXT = FALSE;
   SIGAVG = FALSE;
   InvalidateRect(HWindow, &Area, TRUE);
   }

 if ( DisSumAvg->GetCheck() == BF_CHECKED ) {
   if (!Display) EnableWindow( SumAvgBtn->HWindow, TRUE);
   SetWindowText( HoldBtn->HWindow, "NEXT");
   HOLD = FALSE;
   NEXT = TRUE;
   SIGAVG = TRUE;
   InvalidateRect(HWindow, &Area, TRUE);
   }
}

BOOL ManRxWindow::CanClose()
{
OpenManRx = TRUE;
Display = FALSE;
return 1;
}
```

```
/*******************************************************\
 *
 *       Name: ManTx.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: control the signal pulser card
 *
\*******************************************************/

#define WIN31

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <string.h>
#include <owl.h>
#include <static.h>
#include <edit.h>
#include <listbox.h>
#include <button.h>
#include "mantx.h"

extern BOOL OpenManTx;

ManTxWindow::ManTxWindow(PTWindowsObject AParent) :
   TWindow(AParent, "Manual - Tx")
{
  DisableAutoCreate();
  Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  Attr.X = 5;
  Attr.Y = 70;
  Attr.W = 210;
  Attr.H = 250;

  PW = 250;
  PRF = 60;

  BASEADD = 0x220;

  outp(BASEADD+1, 0x17);
  outp(BASEADD, 0x00);
  outp(BASEADD, 0x00);
  outp(BASEADD+1, 1);
  outp(BASEADD, 0x62);
  outp(BASEADD, 0x0b);
  outp(BASEADD+1, 0xe1);

  ListBox1 = new TListBox(this, ID_MANTXLIST1, 22, 62, 40, 120);
  new TStatic(this, -1, "Pulse", 20, 10, 160, 16, 0);
  new TStatic(this, -1, "Width", 20, 26, 160, 16, 0);
  new TStatic(this, -1, " nsec", 20, 42, 160, 16, 0);
  ListBox2 = new TListBox(this, ID_MANTXLIST2, 84, 30, 50, 180);
  new TStatic(this, -1, "PRF Hz", 80, 10, 160, 16, 0);
  new TButton(this,ID_MANTXBUTT1,"StopTx",140,140,60,30, FALSE);
  new TButton(this,ID_MANTXBUTT2,"Cancel",140,180,60,30, FALSE);
}

void ManTxWindow::SetupWindow()
{
char item[5];
int loop;
  TWindow::SetupWindow();
  for (loop = 0; loop<3; loop++) {
   sprintf(item,"0%i", ((loop+1)*250));
   ListBox1->AddString(item);
  }
  for (loop = 3; loop<6; loop++) {
   sprintf(item,"%i", ((loop+1)*250));
   ListBox1->AddString(item);
  }
  ListBox2->AddString("00060");
  ListBox2->AddString("00090");
  ListBox2->AddString("00125");
  ListBox2->AddString("00220");
  ListBox2->AddString("00520");
  ListBox2->AddString("01000");
  ListBox2->AddString("02020");
  ListBox2->AddString("05000");
  ListBox2->AddString("10000");
};


void ManTxWindow::HandleListBox1Msg(RTMessage Msg)
{
char SelString[5];


  if ( Msg.LP.Hi == LBN_SELCHANGE )
  {
    ListBox1->GetSelString(SelString, sizeof(SelString));
    PW = atoi(SelString);
    StartTx();
  }
}

void ManTxWindow::HandleListBox2Msg(RTMessage Msg)
{
char SelString[7];

  if ( Msg.LP.Hi == LBN_SELCHANGE )
  {
    ListBox2->GetSelString(SelString, sizeof(SelString));
    PRF = atoi(SelString);
    StartTx();
  }
}

void ManTxWindow::HandleButton1Msg(RTMessage)
{
    /* DISARM COUNTER */
    outp(BASEADD+1, 0xc1);
    outp(BASEADD+1, 0x41);
    outp(BASEADD+1, 0x41);
    /* END */
}

void ManTxWindow::StartTx()
{
  float ActualPW = PW * 1e-9;
  float MinWidth = 250e-9;
  unsigned iPrf, iPw;
  float PRFNear;
  float FreqStep = (2e6 - 60.0) / 65534.0;

    PRFNear = ceil(PRF/FreqStep) * FreqStep;
    iPw = floor(PW/250.0);
    iPrf = (unsigned)floor(((1/PRFNear) - ActualPW)/MinWidth);

    /* DISARM COUNTER */
    outp(BASEADD+1, 0xc1);
    outp(BASEADD+1, 0x41);
    outp(BASEADD+1, 0x41);
    /* END */

    outp(BASEADD+1, 0xe1);
    outp(BASEADD+1, 0x09);
    outp(BASEADD, (char)iPw);
    outp(BASEADD, (char)(iPw/256));

    outp(BASEADD+1, 0x11);
    outp(BASEADD, (char)iPrf);
    outp(BASEADD, (char)(iPrf/256));

    outp(BASEADD+1, 0x61);
}

void ManTxWindow::HandleButton2Msg(RTMessage)
{
  outp(BASEADD+1, 0xc1);
  outp(BASEADD+1, 0x41);
  outp(BASEADD+1, 0x41);
  CloseWindow();
  OpenManTx = TRUE;
}

BOOL ManTxWindow::CanClose()
{
 OpenManTx = TRUE;
 return 1;
}
```

```
/*******************************************************\
 *
 *       Name: ManMux.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: control the multiplexer card
 *
\*******************************************************/

#define WIN31

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <string.h>
#include <owl.h>
#include <static.h>
#include <edit.h>
#include <listbox.h>
#include <button.h>
#include "manmux.h"

extern BOOL OpenManMux;

ManMuxWindow::ManMuxWindow(PTWindowsObject AParent) :
   TWindow(AParent, "Manual - MUX")
{
  DisableAutoCreate();
  Attr.Style |= WS_POPUPWINDOW | WS_CAPTION;
  Attr.X = 5;
  Attr.Y = 350;
  Attr.W = 210;
  Attr.H = 290;

  MUX = 0;
  BAOFF = 0;
  BABIT = 0;
  BASEADD = 0x200;
  BITSTATE = 0;

  for (int loop = 0; loop<2; loop++) {
   outp(BASEADD+loop, 0);
  };

  for (int loopA = 4; loopA<6; loopA++) {
   outp(BASEADD+loopA, 0);
  };


  ListBox = new TListBox(this, ID_LISTBOX, 20, 34, 60, 180);
  new TButton(this,ID_BUTTON1,"TOGGLE",100,80,80,30,TRUE);
  new TButton(this,ID_BUTTON2,"Cancel",130,220,60,30,FALSE);
  new TButton(this,ID_BUTTON3,"All Open",100,120,80,30,TRUE);
  new TButton(this,ID_BUTTON4,"All Close",100,160,80,30,TRUE);
  Edit = new TEdit(this,ID_EDIT, "",100,34,100,30,20,FALSE);
  new TStatic(this, -1, "Select:", 20, 10, 160, 20, 0);
  new TStatic(this, -1, "Mux:", 100, 10, 160, 20, 0);
}

void ManMuxWindow::SetupWindow()
{
char item[4];
int loop;
  TWindow::SetupWindow();

  for (loop = 0; loop<10; loop++) {
   sprintf(item,"0%i", loop);
   ListBox->AddString(item);
  }
  for (loop = 10; loop<32; loop++) {
   sprintf(item,"%i", loop);
   ListBox->AddString(item);
  }

  ListBox->SetSelIndex(0);

  FillEdit();
};


void ManMuxWindow::HandleListBoxMsg(RTMessage Msg)
{
char SelString[4];
div_t baoffset;
int readport;
int checkon;

 if ( Msg.LP.Hi == LBN_SELCHANGE )
  {
  ListBox->GetSelString(SelString, sizeof(SelString));
  MUX = atoi(SelString);
  baoffset = div(MUX, 8);
  BAOFF = baoffset.quot;
  BABIT = baoffset.rem;
  if (BAOFF == 2) BAOFF = 4;
  if (BAOFF == 3) BAOFF = 5;
  readport = inp(BASEADD+BAOFF);
  checkon = (readport & (1 << BABIT));
  BITSTATE = checkon >> BABIT;

  FillEdit();
  }
}

void ManMuxWindow::HandleButton1Msg(RTMessage)
{
int readport;
int toggle;

 readport = inp(BASEADD+BAOFF);

 if (BITSTATE) toggle = (readport ^ (1 << BABIT));
 else toggle = (readport | (1 << BABIT));
 outp(BASEADD+BAOFF, toggle);
 BITSTATE = ~BITSTATE;

 FillEdit();
}

void ManMuxWindow::HandleButton2Msg(RTMessage)
{
  CloseWindow();
  OpenManMux = TRUE;
}

BOOL ManMuxWindow::CanClose()
{
  OpenManMux = TRUE;
  return 1;
}

void ManMuxWindow::HandleButton3Msg(RTMessage)
{
int loopA, loopB;

  for (loopA = 0; loopA<=1; loopA++) {
   outp(BASEADD+loopA, 0);
  };
  for (loopB = 4; loopB<=5; loopB++) {
   outp(BASEADD+loopB, 0);
  };
  FillEdit();
}


void ManMuxWindow::HandleButton4Msg(RTMessage)
{
int loopA, loopB;

  for (loopA = 0; loopA<=1; loopA++) {
   outp(BASEADD+loopA, 255);
  };
  for (loopB = 4; loopB<=5; loopB++) {
   outp(BASEADD+loopB, 255);
  };
  FillEdit();
}

void ManMuxWindow::FillEdit()
{
int readport;
int checkon;
char Text[30];

  readport = inp(BASEADD+BAOFF);
  checkon = (readport & (1 << BABIT));
  checkon = checkon >> BABIT;

  if (checkon) {
   sprintf(Text,"%i%s", MUX , " Closed");
  }
  else if (!checkon) {
   sprintf(Text,"%i%s", MUX , " Open");
  };

  Edit->SetText(Text);
}
```

```
/*******************************************************\
 *
 *       Name: ManRx.h
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: data structures for receiver card
 *
\*******************************************************/

#ifndef __MANRX_H
#define __MANRX_H

#ifndef __OWL_H
#include <owl.h>
#endif

#ifndef __LISTBOX_H
#include <listbox.h>
#endif

#ifndef __EDIT_H
#include <edit.h>
#endif

#ifndef __BUTTON_H
#include <button.h>
#endif

#ifndef __SCROLLBA_H
#include <scrollba.h>
#endif

#ifndef __RADIOBUT_H
#include <radiobut.h>
#endif

#ifndef __GROUPBOX_H
#include <groupbox.h>
#endif

#define ID_MANRXBUTT1 321
#define ID_MANRXBUTT2 322
#define ID_MANRXBUTT3 323
#define ID_MANRXBUTT4 324
#define ID_MANRXBUTT5 325
#define ID_MANRXBUTT6 326
#define ID_MANRXBUTT7 327
#define ID_MANRXBUTT8 328
#define ID_MANRXBUTT9 329
#define ID_MANRXBUTT10 340
#define ID_MANRXBUTT11 341
#define ID_MANRXBUTT12 342
#define ID_MANRXBUTT13 343
#define ID_MANRXBUTT14 344
#define ID_MANRXBUTT15 345
#define ID_MANRXBUTT16 346
#define ID_MANRXBUTT17 347
#define ID_MANRXBUTT18 348
#define ID_MANRXBUTT19 349
#define ID_MANRXBUTT20 350
#define ID_MANRXBUTT21 351
#define ID_MANRXBUTT22 352
#define ID_MANRXBUTT23 353
#define ID_MANRXBUTT24 354
#define ID_MANRXBUTT25 355
#define ID_REFRESH     356
#define ID_MANRXZOOM   357
#define ID_MANRXDISRAN 358
#define ID_MANRXDMODE  359
#define ID_MANRXDLIVE  360
#define ID_MANRXDSAVG  361



#define ID_MANRXGATE 390
#define ID_MANRXSCAN 391
#define ID_MANRXTRIG 392



_CLASSDEF(ManRxWindow)
class _EXPORT ManRxWindow : public TWindow
{
public:
  int PW;
  int PRF;
  int BASEADD;

  PTEdit GateEd;
  PTEdit ScanEd;
  PTEdit TrigEd;

  PTButton DisplayBut;
  PTScrollBar ZoomRange;
  PTScrollBar DisplayRange;

  PTGroupBox DisplayMode;
  PTRadioButton DisLive;
  PTRadioButton DisSumAvg;


  virtual BOOL CanClose();
  virtual void DefaultSettings(struct boardparam *);
  virtual void CalImageArea();
  virtual void MeasureGates();

  virtual void Paint(HDC PaintDC, PAINTSTRUCT _FAR &PaintInfo);

  virtual void HandleButton1Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT1];
  virtual void HandleButton2Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT2];
  virtual void HandleButton3Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT3];
  virtual void HandleButton4Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT4];
   virtual void HandleButton5Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT5];
  virtual void HandleButton6Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT6];
   virtual void HandleButton7Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT7];
  virtual void HandleButton8Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT8];
   virtual void HandleButton9Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT9];
  virtual void HandleButton10Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT10];
   virtual void HandleButton11Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT11];
  virtual void HandleButton12Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT12];

  virtual void HandleButton13Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT13];
  virtual void HandleButton14Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT14];
  virtual void HandleButton15Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT15];
  virtual void HandleButton16Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT16];
   virtual void HandleButton17Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT17];
  virtual void HandleButton18Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT18];
   virtual void HandleButton19Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT19];
  virtual void HandleButton20Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT20];
  virtual void HandleButton21Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT21];
   virtual void HandleButton22Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT22];
  virtual void HandleButton23Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT23];
  virtual void HandleButton24Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT24];
  virtual void HandleButton25Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXBUTT25];
  virtual void HandleZoomBarMsg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXZOOM];
  virtual void HandleDisRanBarMsg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXDISRAN];
  virtual void HandleDisModeMsg(RTMessage Msg) =
    [ID_FIRST + ID_MANRXDMODE];
  ManRxWindow(PTWindowsObject AParent);
  virtual void SetupWindow();

};

#endif
```

```
/*******************************************************\
 *
 *       Name: ManTx.h
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: data structures for signal pulser card
 *
\*******************************************************/

#ifndef __MANTX_H
#define __MANTX_H

#ifndef __OWL_H
#include <owl.h>
#endif

#ifndef __LISTBOX_H
#include <listbox.h>
#endif

#ifndef __EDIT_H
#include <edit.h>
#endif

#define ID_MANTXLIST1 311
#define ID_MANTXLIST2 312
#define ID_MANTXBUTT1 313
#define ID_MANTXBUTT2 314
#define ID_MANTXEDIT  315

_CLASSDEF(ManTxWindow)
class _EXPORT ManTxWindow : public TWindow
{
public:
  int PW;
  int PRF;
  int BASEADD;

  PTListBox ListBox1;
  PTListBox ListBox2;
  //PTEdit Edit;
  virtual BOOL CanClose();
  virtual void HandleListBox1Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANTXLIST1];
  virtual void HandleListBox2Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANTXLIST2];
  virtual void HandleButton1Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANTXBUTT1];
  virtual void HandleButton2Msg(RTMessage Msg) =
    [ID_FIRST + ID_MANTXBUTT2];
  ManTxWindow(PTWindowsObject AParent);
  virtual void SetupWindow();
  //virtual void FillEdit(char* Text);
  virtual void StartTx();
};

#endif
```

```
/*******************************************************\
 *
 *       Name: ManMux.h
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: April 1995
 *
 *    Version: 1.0
 *
 * Descrition: data structures for the multiplexer card
 *
\*******************************************************/

#ifndef __MANMUX_H
#define __MANMUX_H

#ifndef __OWL_H
#include <owl.h>
#endif

#ifndef __LISTBOX_H
#include <listbox.h>
#endif

#ifndef __EDIT_H
#include <edit.h>
#endif

#define ID_LISTBOX  101
#define ID_BUTTON1  102
#define ID_BUTTON2  103
#define ID_BUTTON3  104
#define ID_BUTTON4  105
#define ID_EDIT     106


_CLASSDEF(ManMuxWindow)
class _EXPORT ManMuxWindow : public TWindow
{
public:
  int MUX;
  int BAOFF;
  int BABIT;
  int BASEADD;
  int BITSTATE;
  PTListBox ListBox;
  PTEdit Edit;
  ManMuxWindow(PTWindowsObject AParent);
  virtual BOOL CanClose();
  virtual void SetupWindow();
  virtual void HandleListBoxMsg(RTMessage Msg) =
    [ID_FIRST + ID_LISTBOX];
  virtual void HandleButton1Msg(RTMessage Msg) =
    [ID_FIRST + ID_BUTTON1];
  virtual void HandleButton2Msg(RTMessage Msg) =
    [ID_FIRST + ID_BUTTON2];
  virtual void HandleButton3Msg(RTMessage Msg) =
    [ID_FIRST + ID_BUTTON3];
  virtual void HandleButton4Msg(RTMessage Msg) =
    [ID_FIRST + ID_BUTTON4];
  virtual void FillEdit();
};

#endif
```

Version 2: Modified Instrumentation

The second version of the software is given as one program file, called tomo.c

```
/*******************************************************\
 *
 *       Name: Tomo.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: October 1995
 *
 *    Version: 1.0
 *
 * Descrition: main program routine
 *
\*******************************************************/

#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <fcntl.h>
#include <dos.h>
#include "c:\boards\at-gpib\c\decl.h"

#define MAXSWEEP 1
#define MAXCHAN  4

int  nicolet;
int  keithley = 0x200;
int gpib;
char MsgText[100];
int MUX;
int RX;
int DEBUG = 0;
int MaxRx = 0;
int MaxTx = 0;
int RxSide[8];
int TxSide[2][32];
int chanloop = 1;
int sweeploop = 1;

////////////////////////////////

 void finderr();
 void error();
 void dvmerr();
 void filerr(char* File);
 void write_nicolet();
 void NicoletSetup();
 void NicoletCollect(char* FileName);
 void KeithleySetup();
 void KeithleyClose();
 void KeithleyOpen();
 void ReadConfiguration();

////////////////////////////////

void main() {
int muxloop, rxloop;
char temp;
char fname[80];
char BaseName[6];
char DirName[50];

 ReadConfiguration();

 if ((gpib = ibfind("GPIB0")) < 0) error();

 KeithleySetup();
 NicoletSetup();

 printf("Enter base name for waveform *.dat files
                                    (upto 5 characters) > ");
 scanf("%s",&BaseName);
 printf("Enter sub-directory of c:\\waveform\\ > ");
 scanf("%s",&DirName);

 for (rxloop = 0; rxloop<2; rxloop++) {
  printf("\a** Connect Receivers %i to %i then press a key **\n",
       ((rxloop*4)+1), ((rxloop+1)*4) );
  scanf("%s",&temp);
  RX = rxloop;
  for (muxloop = 0; muxloop<MaxTx; muxloop++) {
   MUX = muxloop;
   sprintf(fname,"c:\\waveform\\%s\\%s", DirName,BaseName);
   KeithleyClose();
   NicoletCollect(fname);
   KeithleyOpen();
  }
  printf("\n\n");
 }
}

////////////////////////////////

void ReadConfiguration()
{
 FILE *config;
 int loop;

 config = fopen("c:\\software\\nicolet\\config","r");
 if (config == NULL) filerr("config");
 printf("Reading configuration file\n");

 fscanf(config, "%i", &MaxRx);
 fscanf(config, "%i", &MaxTx);
 for (loop=0; loop<MaxRx; loop++) {
  fscanf(config, "%i", &RxSide[loop]);
 }
 for (loop=0; loop<MaxTx; loop++) {
  fscanf(config, "%i %i", &TxSide[0][loop], &TxSide[1][loop]);
 }
 fclose(config);
 printf("Maximum number of Receivers: %i\n", MaxRx);
 printf("Maximum number of Transmitters: %i\n", MaxTx);
 for (loop=0; loop<MaxRx; loop++) {
  printf("Reciever %i is located on side %i\n",
        (loop+1), RxSide[loop]);
 }
}

////////////////////////////////

void NicoletSetup()
{
// get nicolet
   if ((nicolet = ibfind("DEV14")) < 0) finderr();

// indicate link by beep
   printf("Connected to Nicolet\n");
   sprintf(MsgText,":BEEP");
   write_nicolet();
   sprintf(MsgText,":BEEP");
   write_nicolet();

// setup trigger
   printf("initialising trigger\n");
   sprintf(MsgText,":TRIGGER:MODE NORM");
   write_nicolet();
   sprintf(MsgText,":TRIGGER:SOURCE EXTERNAL");
   write_nicolet();
   sprintf(MsgText,
     ":EXTERNAL:TRIGGER:SETUP ONBOARD,POS,0.512,0.460");
   write_nicolet();


// setup channels
   printf("initialising channels\n");
   sprintf(MsgText,":CHANNELS:DELAY 0.00000495");
   write_nicolet();
   sprintf(MsgText,":CHANNELS:DISPLAY ON");
   write_nicolet();
   sprintf(MsgText,":CHANNELS:FILTER OFF");
   write_nicolet();
   sprintf(MsgText,":CHANNELS:LENGTH 2000");
   write_nicolet();
   sprintf(MsgText,":CHANNELS:REFERENCE OFF");
   write_nicolet();
   sprintf(MsgText,":CHANNELS:REFPOS 0");
   write_nicolet();
   sprintf(MsgText,":CHANNELS:COLLECT ON");
   write_nicolet();
}

void NicoletCollect(char* FileName)
{
char* ReadCharBit;
char* ReadCharByte;
char ReadBuffer[128];
char* ReadTest;
int byte_length;
int byte_count;
int bcount, saveloop;
int numcount;
float voltage;
```

```
float volt;
double vnorm;
double vzero;
double vusroffset;
double vusrscale;
FILE *fp;
char RealFileName[150];
int fileindex;
char look[15];
char* compare = "HOLDNEXT";
int checkit;

// activate signal averaging

   printf("Collecting waveforms\n");
   sprintf(MsgText,":FUNCTION:SUMAVG ON,20,OFF,OFF,OFF");
   write_nicolet();
   sprintf(MsgText,":STORAGE HOLDNEXT");
   write_nicolet();

// collect four waveforms

   do {
    sprintf(MsgText,":STORAGE?");
    write_nicolet();
    ibrd(nicolet, ReadTest, 10);
    } while( (strncmp(ReadTest, compare, 8)) == 0 );

   printf("Waveforms collected ... transfering data\n");

// transfer waveform data

   for ( chanloop=1; chanloop<=MAXCHAN; chanloop++) {

// open ibm file

   fileindex = ( (RX*(MaxTx*4)) + ((MUX)*4) + chanloop );
   if (fileindex<10) sprintf(RealFileName, "%s00%i.dat",
                FileName, fileindex);
   else if ( (fileindex>=10) && (fileindex<100) )
        sprintf(RealFileName,"%s0%i.dat",FileName,fileindex);
   else sprintf(RealFileName,"%s%i.dat",FileName,fileindex);

// check that receiver is on a different side to the transmitter
    // ||
    if (  (TxSide[0][MUX] != RxSide[((RX*4)+(chanloop-1))]) &&
  (TxSide[1][MUX] != RxSide[((RX*4)+(chanloop-1))]) ) {

    fp = fopen(RealFileName, "wt");
    if (fp == NULL) filerr(RealFileName);

// gets data

  printf("Channel %i:\n",chanloop);
  sprintf(MsgText,":WAVEFORM:SOURCE CHANNEL%i,TIME", chanloop);
  write_nicolet();
  sprintf(MsgText,":DISPLAY:YTSELECT CHANNEL%i",chanloop);
  write_nicolet();

  sprintf(MsgText,":WAVEFORM:VNORM?");
  write_nicolet();
  ibrd(nicolet, ReadTest, 20);
  vnorm = atof(ReadTest);
  if (DEBUG) printf("VNorm is %f\n", vnorm);

  sprintf(MsgText,":WAVEFORM:VZERO?");
  write_nicolet();
  ibrd(nicolet, ReadTest, 20);
  vzero = atof(ReadTest);
  if (DEBUG) printf("VZero is %f\n", vzero);

  sprintf(MsgText,":WAVEFORM:YOFFSET?");
  write_nicolet();
  ibrd(nicolet, ReadTest, 20);
  vusroffset = atof(ReadTest);
  if (DEBUG) printf("VUserOffset is %f\n", vusroffset);

  sprintf(MsgText,":WAVEFORM:YSCALE?");
  write_nicolet();
  ibrd(nicolet, ReadTest, 20);
  vusrscale = atof(ReadTest);
  if (DEBUG) printf("VUserScale is %f\n", vusrscale);

  sprintf(MsgText,":WAVEFORM:FORMAT 16,BHL");
  write_nicolet();
  sprintf(MsgText,":WAVEFORM:DATA? 0,%i,1",2000);
  write_nicolet();
  sprintf(ReadCharBit,"a");
  while ( (strncmp(ReadCharBit, "#", 1))!=0 )
        ibrd(nicolet, ReadCharBit, 1);
  ibrd(nicolet, ReadCharBit, 1);
  byte_length = atoi(ReadCharBit); // 8bits
  ibrd(nicolet, ReadCharByte, 9);
  byte_count = atoi(ReadCharByte);

  if (DEBUG) printf("byte length = %i, byte count = %i\n",
                    byte_length, byte_count);

// save data

 printf("saving \"%s\" to disk\n", RealFileName);
 numcount=0;
 for (bcount = 0; bcount < byte_count; bcount+=128) {
  ibrd(nicolet, ReadBuffer, 128);
  for (saveloop=0; saveloop<128; saveloop+=2) {
   numcount++;
   volt=(float)(ReadBuffer[saveloop]
             +(256*ReadBuffer[saveloop+1]));
   if (volt>32767) volt=-(65536-volt);
   voltage=(float)((((volt-vzero)*vnorm)*vusrscale)
             +vusroffset);
   if (numcount<=2000) fprintf(fp,"%f\n",voltage);
   }
   printf(".");
   }
   printf("\n");
   fclose(fp);
  }
  else printf("Channel %i: data ignored\n",chanloop);
 }
}

void KeithleySetup()
{
int loop;

  printf("Connected to Keithley\n");

  for (loop = 0; loop<2; loop++) {
   outp(keithley+loop, 0);
  };

  for (loop = 4; loop<6; loop++) {
   outp(keithley+loop, 0);
  };
}

void KeithleyClose()
{
div_t baoffset;
int BAOFF, BABIT;
int readport, checkon;
int BITSTATE;
int toggle;

  baoffset = div(MUX, 8);
  BAOFF = baoffset.quot;
  BABIT = baoffset.rem;
  if (BAOFF == 2) BAOFF = 4;
  if (BAOFF == 3) BAOFF = 5;
  readport = inp(keithley+BAOFF);
  checkon = (readport & (1 << BABIT));
  BITSTATE = checkon >> BABIT;

  if (!BITSTATE) {
     toggle = (readport | (1 << BABIT));
     outp(keithley+BAOFF, toggle);
  }

  delay(200);

  printf("multiplexer channel %i closed\n", MUX);
}

void KeithleyOpen()
{
div_t baoffset;
int BAOFF, BABIT;
int readport, checkon;
int BITSTATE;
int toggle;

  baoffset = div(MUX, 8);
  BAOFF = baoffset.quot;
  BABIT = baoffset.rem;
  if (BAOFF == 2) BAOFF = 4;
  if (BAOFF == 3) BAOFF = 5;
  readport = inp(keithley+BAOFF);
  checkon = (readport & (1 << BABIT));
  BITSTATE = checkon >> BABIT;

  if (BITSTATE) {
     toggle = (readport ^ (1 << BABIT));
     outp(keithley+BAOFF, toggle);
    }

  printf("multiplexer channel %i opened\n", MUX);
}
```

```
void finderr() {
/* This routine would notify you that the ibfind
   call failed, and refer you to the handler
   software configuration procedures.              */

   printf("Ibfind error;\n");
   exit(1);
}

void error() {
/* An error checking routine at this location would,
   among other things, check iberr to determine the
   exact cause of the error condition and then take
   action appropriate to the application.  For
   errors during data transfers, ibcnt may be
   examined to determine the actual number of bytes
   transferred.                                    */

   printf("GPIB function call error:\n");
   printf("ibsta=0x%x, iberr=0x%x,",ibsta,iberr);
   printf(" ibcnt=0x%x\n",ibcnt);
   exit(1);
}

void dvmerr() {
/* A routine at this location would analyze the fault
   code returned in the DVM's status byte and take
   appropriate action.                             */

   printf("Device error\n");
   printf("ibsta=0x%x, iberr=0x%x,",ibsta,iberr);
   printf(" ibcnt=0x%x\n",ibcnt);

   ibloc(nicolet);

   exit(1);
}

void filerr(char* File) {
/* Cannot open IBM file */

   printf("File error\n");
   printf("cannot open file %s\n", File);
   exit(1);
}

void write_nicolet() {

    ibsic(nicolet);

    ibwrt(nicolet, MsgText, ((strlen(MsgText))+0));
    if (DEBUG) printf("%s\n",MsgText);
    sprintf(MsgText,"");
    if (ibsta & ERR) dvmerr();
}
```

Appendix C

Program Code for the Receiver Arrangement Simulator

The program code for the simulator is divided into five program files and one header file:

us-main.c	is the main program subroutine.
us-err.c	handles error messages and warnings.
us-windows.c	defines the graphical interface routines.
us-rayanal.c	performs the raypath analysis.
us-simset.c	outputs simulator settings.
us.h	header file defining the data classes.

```
/*********************************************************\
 *
 *       Name: Us-Main.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: 1995
 *
 *    Version: 1.0
 *
 * Descrition: main program routine
 *
\*********************************************************/

#include "us.h"
US us;
INPUT input;
USDATA usdata;

int main (argc, argv)
 int argc;
 char **argv;
{
 char *display_name = NULL;
 char *file_basename = NULL;
 int c;

 /* make sure random number seeded at start */
 srandom (c = time (0));

 if (argc > 1)
  file_basename = argv[1];
 else
  file_basename = "testdata";

 us.base_frame = (Frame) xv_create (NULL, FRAME,
   FRAME_LABEL, "Ultrasonic Tomography Simulator",
   NULL);

 PanelCreate (file_basename);
 window_fit (us.base_frame);
 xv_main_loop (us.base_frame);
 return (TRUE);
}
```

```
/*********************************************************\
 *
 *       Name: Us-Err.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: 1995
 *
 *    Version: 1.0
 *
 * Descrition: error messages and warnings
 *
\*********************************************************/

#include "us.h"

int Warning (str, fname)
 char *str;
 char *fname;
{
 /* display error message str to stderr, and continue,
    because a potential error has occured */
 fprintf (stderr, "%s - %s\n", str, fname);
 return (TRUE);
}

int Error (str)
 char *str;
{
 /* display error message str to stderr, and exit,
    because a fatal error has occured */
 fprintf (stderr, "%s\n", str);
 exit (0);
 return (TRUE);
}

int Report (str)
 char *str;
{
 fprintf (stdout, "%s\n", str);
 return (TRUE);
}
```

```
/*******************************************************\
 *
 *       Name: Us-Windows.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: 1995
 *
 *    Version: 1.0
 *
 * Descrition: graphical interface routines
 *
\*******************************************************/

#define XVIEW_DEVID 0x7c

#include "us.h"
extern US us;

extern void RaypathAnalysis ();
extern void SimulatorSettings ();

int PanelCreate (s)
 char *s;
{
 Drawable drawable;
 int width, height;
 Menu menu, menu1, menu2;

 us.panel = xv_find (us.base_frame, PANEL,
        PANEL_LAYOUT, PANEL_VERTICAL,
        NULL);

 menu = (Menu) xv_find (NULL, MENU,
        MENU_NOTIFY_PROC, notify_proc,
        MENU_ACTION_ITEM, "Perform Raypath Analysis",
          RaypathAnalysis,
        MENU_ACTION_ITEM, "Create Training/Testing Data",
          RayDistrib,
        MENU_ACTION_ITEM, "Record Simulator Settings",
          SimulatorSettings,
        NULL);

 us.FileBaseName = xv_find (us.panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "Base name for data files:",
        PANEL_VALUE, s,
        PANEL_VALUE_DISPLAY_LENGTH, 20,
        PANEL_VALUE_STORED_LENGTH, 20,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 20,
        0);

 us.Directory = xv_find (us.panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "File Directory:",
        PANEL_VALUE, "C/Simulator/UltraSound",
        PANEL_VALUE_DISPLAY_LENGTH, 40,
        PANEL_VALUE_STORED_LENGTH, 40,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 20,
        0);

 us.ImageRes = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Image Resolution X by Y:",
        PANEL_VALUE, MAXIMAGERES,
        PANEL_MAX_VALUE, MAXIMAGERES,
        PANEL_MIN_VALUE, 4,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 60,
        0);

 us.VelocityMat = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Material Velocity:",
        PANEL_VALUE, 100,
        PANEL_MAX_VALUE, 100,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 100,
        0);

 us.VelocityDef = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Defect Velocity:",
        PANEL_VALUE, 50,
        PANEL_MAX_VALUE, 100,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 140,
        0);

 us.FrequencyDef = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "% Defect Frequency Scatter:",
        PANEL_VALUE, 50,
        PANEL_MAX_VALUE, 100,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 180,
        0);

 us.GNoise = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Gaussian Noise (0.0 - 1.0):",
        PANEL_VALUE, 50,
        PANEL_MAX_VALUE, 100,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 220,
        0);

 us.MinDefDiameter = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Minimum Defect Diameter:",
        PANEL_VALUE, 1,
        PANEL_MAX_VALUE, 20,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 260,
        0);

 us.MaxDefDiameter = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Maximum Defect Diameter:",
        PANEL_VALUE, MAXDEFECTDIA,
        PANEL_MAX_VALUE, MAXDEFECTDIA,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 300,
        0);

 us.Anisotropy = xv_find (us.panel, PANEL_CYCLE,
        PANEL_LABEL_STRING, "In plane Fibre Direction:",
        PANEL_CHOICE_STRINGS, "Horizontal", "Vertical", 0,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 340,
        0);

 us.PatternType = xv_find (us.panel, PANEL_CYCLE,
        PANEL_LABEL_STRING, "Pattern Type:",
        PANEL_CHOICE_STRINGS, "Pass Defect Thro Scan Area",
                              "Random Defect Positioning", 0,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 380,
        0);

 us.DefectShape = xv_find (us.panel, PANEL_CYCLE,
        PANEL_LABEL_STRING, "Defect Shape:",
        PANEL_CHOICE_STRINGS, "Rectangular", "Circular", 0,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 420,
        0);

 us.SensorRes = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Sensor Resolution:",
        PANEL_VALUE, 10,
        PANEL_MAX_VALUE, 20,
        PANEL_MIN_VALUE, 4,
        PANEL_VALUE_X, 650,
        PANEL_VALUE_Y, 60,
        0);

 us.TomoTopology = xv_find (us.panel, PANEL_CYCLE,
        PANEL_LABEL_STRING, "Tomography Topology:",
        PANEL_NOTIFY_PROC, tomotop,
        PANEL_CHOICE_STRINGS,
          "4 sides to specified set of receivers",
          "3 other sides to all sides receiving",
          "3 sides to only one side of receivers",
          "Opposite sides only", 0,
        PANEL_VALUE_X, 650,
        PANEL_VALUE_Y, 100,
        0);

 us.NumReceivers = xv_find (us.panel, PANEL_SLIDER,
        PANEL_LABEL_STRING, "Number of Receivers:",
        PANEL_VALUE, 16,
        PANEL_MAX_VALUE, 16,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_X, 650,
        PANEL_VALUE_Y, 140,
        0);

 us.ReceiverPos = xv_find (us.panel, PANEL_CYCLE,
        PANEL_LABEL_STRING, "Receiver Positions:",
        PANEL_NOTIFY_PROC, recpos,
        PANEL_CHOICE_STRINGS, "Explicitly Specified",
                              "Randomly Placed", 0,
        PANEL_VALUE_X, 650,
        PANEL_VALUE_Y, 180,
        0);

 us.RecPos1 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 1:",
        PANEL_VALUE, 1,
```

```
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 220,
        0);

us.RecPos2 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 2:",
        PANEL_VALUE, 2,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 250,
        0);

us.RecPos3 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 3:",
        PANEL_VALUE, 3,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 280,
        0);

us.RecPos4 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 4:",
        PANEL_VALUE, 4,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 310,
        0);

us.RecPos5 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 5:",
        PANEL_VALUE, 5,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 340,
        0);

us.RecPos6 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 6:",
        PANEL_VALUE, 6,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 370,
        0);

us.RecPos7 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 7:",
        PANEL_VALUE, 7,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 400,
        0);

us.RecPos8 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 8:",
        PANEL_VALUE, 8,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 550,
        PANEL_VALUE_Y, 430,
        0);

us.RecPos9 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 9:",
        PANEL_VALUE, 9,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 220,
        0);

us.RecPos10 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 10:",
        PANEL_VALUE, 10,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 250,
        0);

us.RecPos11 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 11:",
        PANEL_VALUE, 11,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 280,
        0);

us.RecPos12 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 12:",
        PANEL_VALUE, 12,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 310,
        0);

us.RecPos13 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 13:",
        PANEL_VALUE, 13,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 340,
        0);

us.RecPos14 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 14:",
        PANEL_VALUE, 14,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 370,
        0);

us.RecPos15 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 15:",
        PANEL_VALUE, 15,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 400,
        0);

us.RecPos16 = xv_find (us.panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING, "Rec Pos 16:",
        PANEL_VALUE, 16,
        PANEL_MAX_VALUE, 80,
        PANEL_MIN_VALUE, 1,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE_STORED_LENGTH, 2,
        PANEL_VALUE_X, 800,
        PANEL_VALUE_Y, 430,
        0);

xv_find (us.panel, PANEL_BUTTON,
        PANEL_LABEL_STRING, "MAIN MENU",
        PANEL_ITEM_MENU, menu,
        PANEL_VALUE_X, 220,
        PANEL_VALUE_Y, 460,
        NULL);

xv_find (us.panel, PANEL_BUTTON,
        PANEL_LABEL_STRING, "Push to quit",
        PANEL_NOTIFY_PROC, quit,
        PANEL_VALUE_X, 850,
        PANEL_VALUE_Y, 460,
        0);

window_fit (us.panel);
```

```
}

int event_proc (window, event)
 Xv_Window window;
 Event *event;
{
 if (event_action (event) == ACTION_MENU
    && event_is_down (event)) {
  Menu menu = (Menu) xv_get (window, WIN_CLIENT_DATA);
  menu_show (menu, window, event, NULL);
 };
}

int somerecon ()
{
 int some;

 some = xv_get (us.NumReceivers, PANEL_VALUE);
 if (some>=1) xv_set(us.RecPos1,PANEL_INACTIVE,FALSE,NULL);
 if (some>=2) xv_set(us.RecPos2,PANEL_INACTIVE,FALSE,NULL);
 if (some>=3) xv_set(us.RecPos3,PANEL_INACTIVE,FALSE,NULL);
 if (some>=4) xv_set(us.RecPos4,PANEL_INACTIVE,FALSE,NULL);
 if (some>=5) xv_set(us.RecPos5,PANEL_INACTIVE,FALSE,NULL);
 if (some>=6) xv_set(us.RecPos6,PANEL_INACTIVE,FALSE,NULL);
 if (some>=7) xv_set(us.RecPos7,PANEL_INACTIVE,FALSE,NULL);
 if (some>=8) xv_set(us.RecPos8,PANEL_INACTIVE,FALSE,NULL);
 if (some>=9) xv_set(us.RecPos9,PANEL_INACTIVE,FALSE,NULL);
 if (some>=10) xv_set(us.RecPos10,PANEL_INACTIVE,FALSE,NULL);
 if (some>=11) xv_set(us.RecPos11,PANEL_INACTIVE,FALSE,NULL);
 if (some>=12) xv_set(us.RecPos12,PANEL_INACTIVE,FALSE,NULL);
 if (some>=13) xv_set(us.RecPos13,PANEL_INACTIVE,FALSE,NULL);
 if (some>=14) xv_set(us.RecPos14,PANEL_INACTIVE,FALSE,NULL);
 if (some>=15) xv_set(us.RecPos15,PANEL_INACTIVE,FALSE,NULL);
 if (some>=16) xv_set(us.RecPos16,PANEL_INACTIVE,FALSE,NULL);
}

int allrecoff ()
{
 xv_set (us.RecPos1, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos2, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos3, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos4, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos5, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos6, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos7, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos8, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos9, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos10, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos11, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos12, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos13, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos14, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos15, PANEL_INACTIVE, TRUE, NULL);
 xv_set (us.RecPos16, PANEL_INACTIVE, TRUE, NULL);
}

int tomotop (item, event)
 Panel_item item;
 Event *event;
{
 int val;

 val = xv_get (us.TomoTopology, PANEL_VALUE);
 switch (val) {
 case 0:
  xv_set (us.ReceiverPos, PANEL_INACTIVE, FALSE, NULL);
  if ((xv_get (us.ReceiverPos, PANEL_VALUE)) == 0) {
   allrecoff ();
   somerecon ();
  }
  break;
 case 1:
  xv_set (us.ReceiverPos, PANEL_INACTIVE, TRUE, NULL);
  allrecoff ();
  break;
 case 2:
  xv_set (us.ReceiverPos, PANEL_INACTIVE, TRUE, NULL);
  allrecoff ();
  break;
 case 3:
  xv_set (us.ReceiverPos, PANEL_INACTIVE, TRUE, NULL);
  allrecoff ();
  break;
 default:
  xv_set (us.ReceiverPos, PANEL_INACTIVE, FALSE, NULL);
  if ((xv_get (us.ReceiverPos, PANEL_VALUE)) == 0) {
   allrecoff ();
   somerecon ();
  }
  break;
 };
}

int recpos (item, event)
 Panel_item item;
 Event *event;
{
 int val;

 val = xv_get (us.ReceiverPos, PANEL_VALUE);
 if (val == 1) allrecoff ();
 if (val == 0) {
  allrecoff ();
  somerecon ();
 }
}
```

```
/*******************************************************\
 *
 *       Name: Us-Rayanal.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: 1995
 *
 *    Version: 1.0
 *
 * Descrition: performs raypath analysis
 *
\*******************************************************/

#include "us.h"
extern US us;
extern INPUT input;
extern USDATA usdata;

FILE *fp, *fpmat;
char File[130];

int SetUpDataStructure ()
{
 int Tx;
 int loop;
 int sloop, dloop, xloop, yloop;
 float scale;
 int countdefectsofsize;
 int totaldefects;
 int halfsloop;

 /* Check for abnormal situations */

 if (input.mindefdia > input.maxdefdia) {
  Report ("Range of defect sizes is unclear, please change");
  return (FALSE);
 }
 Tx = ((4 * input.sensorres) - input.numrec);
 if (Tx == 0) {
  Report ("There are zero transmitters in this configuration\n
           Either increase sensor resolution or decrease number of receivers");
  return (FALSE);
 }

 /* copy user input to data structure */

 usdata.info = input;
 usdata.NumofTxers = Tx;

 /* calculate transducer positions */

 if (DEBUG)
  Report ("Calculating transducer positions ... ");
/* 4 sides to specified set of receivers */
 if (usdata.info.tomotop == 0) {
 Report ("Selected tomographic topology: Specified set of Receivers");

 for (loop = 1; loop < (((usdata.info.sensorres) * 4) + 1); loop++) {
  if (loop < ((usdata.info.sensorres) + 1)) {
   usdata.TxPosX[loop] = SCANSIZE;
   usdata.TxPosY[loop] = ((loop*(int)((SCANSIZE)/(usdata.info.sensorres)))-(int)(0.5*((SCANSIZE)/(usdata.info.sensorres))));
   } else if (loop < ((2 * (usdata.info.sensorres)) + 1)) {
    usdata.TxPosY[loop] = SCANSIZE;
    usdata.TxPosX[loop] = (((((2*(usdata.info.sensorres))+1)-loop)*(int)((SCANSIZE)/(usdata.info.sensorres)))
             -(int)(0.5*((SCANSIZE)/(usdata.info.sensorres))));
   } else if (loop < ((3 * (usdata.info.sensorres)) + 1)) {
    usdata.TxPosX[loop] = 1;
    usdata.TxPosY[loop] = (((((3*(usdata.info.sensorres))+1)-loop)*(int)((SCANSIZE)/(usdata.info.sensorres)))
             -(int) (0.5 * ((SCANSIZE) / (usdata.info.sensorres))));
   } else if (loop < ((4 * (usdata.info.sensorres)) + 1)) {
    usdata.TxPosY[loop] = 1;
    usdata.TxPosX[loop] = (((loop-(3*(usdata.info.sensorres)))*(int)((SCANSIZE)/(usdata.info.sensorres)))
             -(int) (0.5 * ((SCANSIZE) / (usdata.info.sensorres))));
  }
 }

  for (loop = 1; loop < (usdata.info.numrec + 1); loop++) {

   if (usdata.info.recpos[loop] < ((usdata.info.sensorres) + 1)) {
    usdata.RxPosX[loop] = SCANSIZE;
    usdata.RxPosY[loop] = (((usdata.info.recpos[loop])*(int)((SCANSIZE)/(usdata.info.sensorres)))
        - (int) (0.5 * ((SCANSIZE) / (usdata.info.sensorres))));
   } else if (usdata.info.recpos[loop] <
    ((2 *(usdata.info.sensorres))+1)) {
     usdata.RxPosY[loop] = SCANSIZE;
     usdata.RxPosX[loop] = (((((2*(usdata.info.sensorres))+1)-usdata.info.recpos[loop])
        *(int)((SCANSIZE)/(usdata.info.sensorres)))-(int)(0.5*((SCANSIZE)/(usdata.info.sensorres))));
   } else if (usdata.info.recpos[loop] <
    ((3*(usdata.info.sensorres))+1)) {
     usdata.RxPosX[loop] = 1;
     usdata.RxPosY[loop] = (((((3*(usdata.info.sensorres))+1)-usdata.info.recpos[loop])
        *(int)((SCANSIZE)/(usdata.info.sensorres)))-(int)(0.5*((SCANSIZE)/(usdata.info.sensorres))));
   } else if (usdata.info.recpos[loop] <
```

```
    ((4*(usdata.info.sensorres))+1)) {
     usdata.RxPosY[loop] = 1;
     usdata.RxPosX[loop] = (((usdata.info.recpos[loop]
        - (3 * (usdata.info.sensorres)))
        * (int) ((SCANSIZE) / (usdata.info.sensorres)))
        - (int) (0.5 * ((SCANSIZE) / (usdata.info.sensorres))));
    }
    if (DEBUG)
     fprintf (fp, "Receiver %i \tx: %i y: %i\n", loop,
           usdata.RxPosX[loop], usdata.RxPosY[loop]);

    usdata.TxPosX[(usdata.info.recpos[loop])] = 0;
    usdata.TxPosY[(usdata.info.recpos[loop])] = 0;
   }

   if (DEBUG)
    for (loop = 1; loop < (((usdata.info.sensorres) * 4) + 1); loop++)
     fprintf (fp, "Transmitter %i \tx: %i y: %i\n", loop,
              usdata.TxPosX[loop], usdata.TxPosY[loop]);
  } else
   Error ("OTHER TOMOGRAPHIC TOPOLOGIES NOT IMPLEMENTED");

  if (DEBUG)
   Report ("Allocating defects within scan area ...");
  if (usdata.info.pattype == 0) {
   Report ("Passing defect(s) thro scan area");
   usdata.RECT = 1;
   totaldefects = 0;
   for(sloop=usdata.info.mindefdia;
       sloop<((usdata.info.maxdefdia)+1);sloop++){
    usdata.DefSize[sloop] = 0;
    scale = (float)(((float)SCANSIZE)/((float)usdata.info.imageres));
    countdefectsofsize = 0;
    if (sloop > usdata.info.mindefdia)
     totaldefects += usdata.DefSize[(sloop - 1)];

    for (xloop = 0; xloop < ((int) (SCANSIZE / scale) + 1); xloop++) {
     for (yloop = 0; yloop < ((int) (SCANSIZE / scale) + 1); yloop++) {

      dloop = totaldefects+((xloop)*((int)(SCANSIZE/scale)+1)+(yloop));
      countdefectsofsize=((xloop)*((int)(SCANSIZE/scale)+1)+(yloop));

      if (sloop > 1) halfsloop = (sloop + 1);
      else halfsloop = sloop;

      usdata.Def[dloop].xl = xloop * scale - (int) (0.5 * sloop);
      usdata.Def[dloop].xr = xloop * scale + (int) (0.5 * (halfsloop));
      usdata.Def[dloop].yt = yloop * scale - (int) (0.5 * sloop);
      usdata.Def[dloop].yb = yloop * scale + (int) (0.5 * (halfsloop));
      usdata.Def[dloop].cenx = xloop * scale;
      usdata.Def[dloop].ceny = yloop * scale;

      if (usdata.Def[dloop].xl < 1) usdata.Def[dloop].xl = 1;
      if (usdata.Def[dloop].xr>SCANSIZE) usdata.Def[dloop].xr=SCANSIZE;
      if (usdata.Def[dloop].xl>SCANSIZE) usdata.Def[dloop].xl=SCANSIZE;

      if (usdata.Def[dloop].yt < 1) usdata.Def[dloop].yt = 1;
      if (usdata.Def[dloop].yb>SCANSIZE) usdata.Def[dloop].yb=SCANSIZE;
      if (usdata.Def[dloop].yt>SCANSIZE) usdata.Def[dloop].yt=SCANSIZE;

      if (usdata.Def[dloop].xl > usdata.Def[dloop].xr)
          usdata.Def[dloop].xr = usdata.Def[dloop].xl;
      if (usdata.Def[dloop].yt > usdata.Def[dloop].yb)
          usdata.Def[dloop].yb = usdata.Def[dloop].yt;

      if (DEBUG)
       fprintf (fp, "Defect %i \t\txl: %i xr: %i \tyt: %i
                    yb: %i cenx: %i ceny: %i\n",
            dloop, usdata.Def[dloop].xl, usdata.Def[dloop].xr,
            usdata.Def[dloop].yt, usdata.Def[dloop].yb,
            usdata.Def[dloop].cenx, usdata.Def[dloop].ceny);
     }
    }
    usdata.DefSize[sloop] = countdefectsofsize + 1;
    if (DEBUG)
     fprintf (fp, "Defect size %i has %i defects\n",
           sloop, usdata.DefSize[sloop]);
   }
   if (DEBUG)
    fprintf (fp, "Total number of defects: %i\n", (dloop + 1));

   usdata.NumofDefects = dloop;
  } else
   Error ("ONLY PASS DEFECT THRO SCAN AREA IMPLEMENTED");
}

void PerformAnalysis ()
{
 int dloop, rloop, tloop, xloop, yloop, sloop, hloop, xxloop;
 long RaypathHitDefect = 0;
 long RaypathVerticalHit = 0;
 long RaypathHorizontalHit = 0;
 long TotalRaypath = 0;
 float gradientX, gradientY;
```

```
 int x, y;
 int xx, yy, xxprev, yyprev;
 int flaghit;
 int totaldefects;
 int startX, startY, endX, endY;
 float interX, interY;

 long RaypathHits[4 * MAXSENSORRES * MAXRXERS];
 long ReceiverHits[usdata.info.numrec];
 long DefectHits[((((MAXIMAGERES+1)*(MAXIMAGERES+1))+1)*MAXDEFECTDIA)];
 long ImageHits[(MAXIMAGERES + 1) * (MAXIMAGERES + 1)];
 int raypath;

 Report ("Performing Raypath Analysis ... please wait");

 TotalRaypath = usdata.info.numrec * usdata.NumofTxers;

 for (rloop = 1; rloop < (usdata.info.numrec + 1); rloop++) {
  for (tloop=1; tloop < ((4* usdata.info.sensorres)+1); tloop++) {
   raypath = ((rloop - 1) * usdata.info.sensorres * 4) + tloop;
   RaypathHits[raypath] = 0;
  }
  ReceiverHits[(rloop - 1)] = 0;
 }

 for (dloop = 1; dloop < (usdata.NumofDefects + 1); dloop++) {
  DefectHits[(dloop - 1)] = 0;
  for (rloop = 1; rloop < (usdata.info.numrec + 1); rloop++) {
   for (tloop = 1; tloop < ((4 * usdata.info.sensorres) + 1); tloop++) {

    raypath = ((rloop - 1) * usdata.info.sensorres * 4) + tloop;
    if ((usdata.TxPosX[tloop] == 0) && (usdata.TxPosY[tloop] == 0));
                     /* skip if Tx position is a Rx */
    else if ((usdata.TxPosX[tloop] == 1) && (usdata.RxPosX[rloop] == 1));
                     /* skip if Tx is on same side as Rx */
    else if ((usdata.TxPosX[tloop] == usdata.info.imageres)
              && (usdata.RxPosX[rloop] == usdata.info.imageres));
    else if ((usdata.TxPosY[tloop] == 1) && (usdata.RxPosY[rloop] == 1));
    else if ((usdata.TxPosY[tloop] == usdata.info.imageres)
              && (usdata.RxPosY[rloop] == usdata.info.imageres));
    else {
     if ((usdata.TxPosX[tloop] == usdata.RxPosX[rloop])) {
                     /* parallel vertical raypath */
      if ((usdata.Def[dloop].xr >= usdata.TxPosX[tloop])
              && (usdata.Def[dloop].xl <= usdata.TxPosX[tloop])) {
       RaypathVerticalHit++;
       RaypathHits[raypath]++;
       ReceiverHits[(rloop - 1)]++;
       DefectHits[(dloop - 1)]++;
      }
     } else if ((usdata.TxPosY[tloop] == usdata.RxPosY[rloop])) {
                     /* parallel horizontal raypath */
      if ((usdata.Def[dloop].yb >= usdata.TxPosY[tloop])
              && (usdata.Def[dloop].yt <= usdata.TxPosY[tloop])) {
       RaypathHorizontalHit++;
       RaypathHits[raypath]++;
       ReceiverHits[(rloop - 1)]++;
       DefectHits[(dloop - 1)]++;
      }
     } else {
      flaghit = 0;

      if ((usdata.RxPosX[rloop] > usdata.TxPosX[tloop])) {
                     /* start at Tx */
       startX = usdata.TxPosX[tloop];
       startY = usdata.TxPosY[tloop];
       endX = usdata.RxPosX[rloop];
       endY = usdata.RxPosY[rloop];
      } else {
       startX = usdata.RxPosX[rloop];
       startY = usdata.RxPosY[rloop];
       endX = usdata.TxPosX[tloop];
       endY = usdata.TxPosY[tloop];
      }

      gradientX = (float) (((float) endY - (float) startY)
                   / ((float) endX - (float) startX));
      gradientY = (float) (1 / gradientX);

      interX = (float) (startY - (gradientX * (float) startX));
      interY = (float) (startX - (gradientY * (float) startY));

      for (x = startX; x < (endX + 1); x++) {
       yy = (int) (((gradientX * x) + interX));
       if (x > startX) {
yyprev = (int) (((gradientX * (x - 1)) + interX));
       } else yyprev = yy;
       if (yy >= yyprev) {
for (hloop = yyprev; hloop < (yy + 1); hloop++) {
 if ((usdata.Def[dloop].xr >= x) && (usdata.Def[dloop].xl <= x)
               && (usdata.Def[dloop].yb >= hloop)
               && (usdata.Def[dloop].yt <= hloop))
  flaghit = 1;
}
```

```
      } else {
for (hloop = yy; hloop < (yyprev + 1); hloop++) {
 if ((usdata.Def[dloop].xr >= x) && (usdata.Def[dloop].xl <= x)
               && (usdata.Def[dloop].yb >= hloop)
               && (usdata.Def[dloop].yt <= hloop))
  flaghit = 1;
}
       }
      }

      if (flaghit) {
       RaypathHitDefect++;
       RaypathHits[raypath]++;
       ReceiverHits[(rloop - 1)]++;
       DefectHits[(dloop - 1)]++;
      }
     }

    }

   }
  }
 }

 if (!DEBUG) {
  for (rloop = 1; rloop < (usdata.info.numrec + 1); rloop++) {
   for (tloop = 1; tloop < ((4 * usdata.info.sensorres) + 1); tloop++) {
    raypath = ((rloop - 1) * usdata.info.sensorres * 4) + tloop;
    if ((usdata.TxPosX[tloop] == 0) && (usdata.TxPosY[tloop] == 0));
              /* skip if Tx position is a Rx */
    else if ((usdata.TxPosX[tloop] == 1) && (usdata.RxPosX[rloop] == 1));
              /* skip if Tx is on same side as Rx */
    else if ((usdata.TxPosX[tloop] == usdata.info.imageres)
             && (usdata.RxPosX[rloop] == usdata.info.imageres));
    else if ((usdata.TxPosY[tloop] == 1) && (usdata.RxPosY[rloop] == 1));
    else if ((usdata.TxPosY[tloop] == usdata.info.imageres)
             && (usdata.RxPosY[rloop] == usdata.info.imageres));
    else {
     fprintf (fp, "%i \t%i \t%i \t%i\n",
              rloop, tloop, raypath, RaypathHits[raypath]);
    }
   }
   fprintf (fp, "\n");
  }
 }
 for (xloop = 0; xloop < (usdata.info.imageres + 1); xloop++) {
  ImageHits[xloop] = 0;
 }

 for (yloop = 0; yloop < (usdata.info.imageres); yloop++) {
  for (xloop = 0; xloop < (usdata.info.imageres); xloop++) {
   totaldefects = 0;
   for (sloop=usdata.info.mindefdia;
         sloop < ((usdata.info.maxdefdia) + 1); sloop++) {
    if (sloop > usdata.info.mindefdia)
     totaldefects += usdata.DefSize[(sloop - 1)];
    for (dloop = 0; dloop < (usdata.DefSize[sloop]); dloop++)
     if ((xloop == (usdata.Def[dloop].cenx))
           && ((yloop) == (usdata.Def[dloop].ceny)))
      ImageHits[xloop] += DefectHits[(totaldefects + dloop)];
   }
  }
  for (xxloop = 0; xxloop < (usdata.info.imageres); xxloop++) {
   fprintf (fpmat, "%i\t", ImageHits[(xxloop)]);
   ImageHits[(xxloop)] = 0;
  }
  fprintf (fpmat, "\n");
 }
}

void RaypathAnalysis ()
{

 measure_input ();

 /* Write Analysis Results to File */

 sprintf (File, "/home/sunbird/es078/%s/%s.ray",
       input.dirname, input.filename);
 fp = fopen (File, "w");
 if (fp == NULL) Warning ("Unable to Write File", File);
 if (fp != NULL) {
  fprintf (stdout, "Writing Raypath Analysis to file:\n%s\n", File);
  sprintf (File, "/home/sunbird/es078/%s/%s.im",
        input.dirname, input.filename);
  fpmat = fopen (File, "w");
  if (fpmat == NULL) Warning ("Unable to Write File", File);

  if (SetUpDataStructure ()) {
   PerformAnalysis ();
  }
 }
 fclose (fp);
}
```

```
/*******************************************************\
 *
 *       Name: Us-Simset.c
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: 1995
 *
 *    Version: 1.0
 *
 * Descrition: output simulator settings
 *
\*******************************************************/

#include "us.h"
extern US us;
extern INPUT input;

void SimulatorSettings ()
{
 char File[150]:
 FILE *fp;
 int loop;

 measure_input ();
 sprintf (File, "/home/sunbird/es078/%s/%s.set",
        input.dirname, input.filename);
 fp = fopen (File, "w");
 if (fp == NULL) Warning ("Unable to Write File", File);
 if (fp != NULL) {
  if (DEBUG)
   fprintf(stdout,"Saving simulator settings to file:
         \n%s\n",File);
   fprintf(fp,"SIMULATOR SETTINGS\n==================\n\n");
   fprintf(fp,"  Directory and File: ~/%s/%s
         \n  -------------------\n\n",
         input.dirname, input.filename);
   fprintf(fp,"    Image Resolution: %i\n",input.imageres);
   fprintf(fp,"   Sensor Resolution: %i\n",input.sensorres);
   fprintf(fp," Number of Receivers: %i\n",input.numrec);
   fprintf(fp," Defect diameter min: %i \n",input.mindefdia);
   fprintf(fp,"                 max: %i\n",input.maxdefdia);

  fprintf (fp, "     Fibre Direction: ");
  switch (input.anisotropy) {
  case 0:
   fprintf (fp, "Horizontal\n");
   break;
  case 1:
   fprintf (fp, "Vertical\n");
   break;
  }

  fprintf (fp, "        Pattern Type: ");
  switch (input.pattype) {
  case 0:
   fprintf (fp, "Pass Defect Thro Scan Area\n");
   break;
  case 1:
   fprintf (fp, "Random Defect Positioning\n");
   break;
  }

  fprintf (fp, "        Defect shape: ");
  switch (input.defshape) {
  case 0:
   fprintf (fp, "Rectangular\n");
   break;
  case 1:
   fprintf (fp, "Circular\n");
   break:
  }

  fprintf (fp, "Tomographic Topology: ");
  switch (input.tomotop) {
  case 0:
   fprintf (fp, "4 sides to specified set of receivers\n");
   break;
  case 1:
   fprintf (fp, "3 other sides to all sides receiving\n");
   break;
  case 2:
   fprintf (fp, "3 sides to only one side of receivers\n");
   break;
  case 3:
   fprintf (fp, "Opposite sides only\n");
   break;
  }

  if (input.receiverpos == 0) {
   fprintf (fp, "  Receiver Positions: ");
   for (loop = 1; loop < ((input.numrec) + 1); loop++)
    fprintf (fp, "%i ", input.recpos[loop]);
   fprintf (fp, "\n");
  } else {
   fprintf (fp, "  Receiver Positions: Randomly Placed\n");
  }
 }
 fclose (fp);
}
```

```
/*******************************************************\
 *
 *       Name: Us.h
 *
 *     Author: Andrew C. Pardoe
 *
 *       Date: 1995
 *
 *    Version: 1.0
 *
 * Descrition: data structures
 *
\*******************************************************/

#ifndef _us_h
#define _us_h

#include <sys/types.h>
#include <xview/attr.h>
#include <xview/defaults.h>
#include <xview/notice.h>
#include <xview/frame.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/canvas.h>
#include <xview/openmenu.h>
#include <xview/scrollbar.h>
#include <xview/textsw.h>
#include <xview/svrimage.h>
#include <xview/cms.h>
#include <xview/font.h>
#include <xview/xv_xrect.h>
#include <xview/cms.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <rasterfile.h>
#include <math.h>
#include <stdio.h>

#define TRUE 1
#define UNKNOWN -1
#define FALSE 0
#define DEBUG 0
#define CMS_NO 256
#define WINDOW_HT 200
#define WINDOW_WD 200
#define MAXIMAGERES 80
#define MAXDEFECTDIA 20
#define MAXSENSORRES 20
#define MAXTXERS     80
#define MAXRXERS     16
#define SCANSIZE     80

/* types */
typedef char Filename[20];

/* Ultrasound variables, structures etc */

typedef struct {
    char filename[130];
    char dirname[130];
    int imageres;
    int velmat;
    int veldef;
    int freqdef;
    int gnoise;
    int defshape;
    int sensorres;
    int tomotop;
    int numrec;
    int mindefdia;
    int maxdefdia;
    int anisotropy;
    int pattype;
    int receiverpos;
    int recpos[17];
} INPUT;

typedef struct {
    int xl;
    int xr;
    int yt;
    int yb;
    int cenx;
    int ceny;
    int dia;
} DEF;

typedef struct {
    INPUT info; /* copy of the user inputs at Initialisation
                   of this data structure */
    int RxPosX[17];
    int RxPosY[17];
    int TxPosX[((4*MAXSENSORRES)+1)];
    int TxPosY[((4*MAXSENSORRES)+1)];
    int NumofTxers;
    int RECT;
    DEF Def[((((MAXIMAGERES+MAXDEFECTDIA)
               *(MAXIMAGERES+MAXDEFECTDIA))+1)
               *MAXDEFECTDIA)];
    int DefSize[(MAXDEFECTDIA+1)];
    int NumofDefects;
} USDATA;

/* Sunview/XView/XLib variables, structures etc */
/* all globals begin in structure of type us */
typedef struct {
  Frame base_frame;
  Panel panel;
  Panel_item FileBaseName;
  Panel_item Directory;
  Panel_item ImageRes;
  Panel_item VelocityMat;
  Panel_item VelocityDef;
  Panel_item FrequencyDef;
  Panel_item GNoise;
  Panel_item DefectShape;
  Panel_item SensorRes;
  Panel_item TomoTopology;
  Panel_item NumReceivers;
  Panel_item MinDefDiameter;
  Panel_item MaxDefDiameter;
  Panel_item Anisotropy;
  Panel_item PatternType;
  Panel_item ReceiverPos;
  Panel_item RecPos1;
  Panel_item RecPos2;
  Panel_item RecPos3;
  Panel_item RecPos4;
  Panel_item RecPos5;
  Panel_item RecPos6;
  Panel_item RecPos7;
  Panel_item RecPos8;
  Panel_item RecPos9;
  Panel_item RecPos10;
  Panel_item RecPos11;
  Panel_item RecPos12;
  Panel_item RecPos13;
  Panel_item RecPos14;
  Panel_item RecPos15;
  Panel_item RecPos16;
} US;

/* us_main.c */
int main();

/* us_err.c */
int Error();
int Warning();
int Reaport();

/* procedures in US package us.c */
int quit();
void notify_proc();
void measure_input();

/* us_rayanal.c */
int SetUpDataStructure();
void RaypathAnalysis();

/* us_simset.c */
void SimulatorSettings();

/* us_windows.c */
int PanelCreate();
int event_proc();
int tomotop();
int recpos();
int allrecoff();
int somerecon();

/* putsuffix */
putsuffix();

#endif _us_h
```

Appendix D

Definition and Derivation of the Principal Components

The following definition and derivation is based on that given by Bishop [123], although an equally comprehensive alternative derivation is given by Jolliffe [103].

Given a data set $\mathbf{Z}$ which contains I variables over P patterns, where a single pattern is represented by vector $\mathbf{z}$. A transformation from data $\mathbf{z}$ to a set of principal components, $\mathbf{u}$, may be defined as shown:

$$\begin{aligned}
u_1 &= \Phi_{11}z_1 + \Phi_{12}z_2 + \cdots + \Phi_{1I}z_I = \sum_{j=1}^{I} \Phi_{1j}z_j \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
u_i &= \Phi_{i1}z_1 + \Phi_{i2}z_2 + \cdots + \Phi_{iI}z_I = \sum_{j=1}^{I} \Phi_{ij}z_j \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
u_I &= \Phi_{I1}z_1 + \Phi_{I2}z_2 + \cdots + \Phi_{II}z_I = \sum_{j=1}^{I} \Phi_{Ij}z_j
\end{aligned} \tag{D.1}$$

where $\mathbf{\Phi}$ is a matrix of numerical parameters.

The above transformation can be written in vector form,

$$u_i = \mathbf{\Phi}_i^\top \mathbf{z}, \tag{D.2}$$

and can be regarded as a simple rotation of the coordinate system if and only if the $\mathbf{\Phi}$ vectors satisfy orthonormality,

$$\mathbf{\Phi}_a^\top \mathbf{\Phi}_b = \begin{cases} 1 \text{ if } a = b \\ 0 \text{ otherwise} \end{cases} \tag{D.3}$$

And so the construction of data $\mathbf{z}$ may be formed from the combination of all the principal components,

$$\mathbf{z} = \sum_{n=1}^{I} u_n \mathbf{\Phi}_n. \tag{D.4}$$

For dimensionality reduction the number of principal components is to be less than the dimensionality of the original data. The error caused by reducing the dimensionality can be evaluated. Assume that the number of principal components is M where $M < I$ and the unused $I - M$ principal components are replaced by parameters q_n. Then the approximation of data $\mathbf{z}$ is given by $\tilde{\mathbf{z}}$ as follows

$$\tilde{\mathbf{z}} = \sum_{n=1}^{M} u_n \mathbf{\Phi}_n + \sum_{n=M+1}^{I} q_n \mathbf{\Phi}_n. \tag{D.5}$$

Thus the difference between the original data $\mathbf{z}$ and the reconstructed data $\tilde{\mathbf{z}}$ when using only M components rather than I is given by

$$\mathbf{z} - \tilde{\mathbf{z}} = \sum_{n=M+1}^{I} (u_n - q_n) \mathbf{\Phi}_n \tag{D.6}$$

Reducing the error in a least squared way, over the entire data set, will give

$$\begin{aligned} \varepsilon &= \frac{1}{2} \sum_{p=1}^{P} \| \mathbf{z}^p - \tilde{\mathbf{z}}^p \|^2 \\ &= \frac{1}{2} \sum_{p=1}^{P} \sum_{n=1}^{I} \left(u_n^p - q_n \right)^2, \end{aligned} \tag{D.7}$$

with q_n defined as

$$q_n = \frac{1}{P}\sum_{p=1}^{P} u_n = \Phi^{\top}\bar{\mathbf{z}} \tag{D.8}$$

where $\bar{\mathbf{z}}$ is the mean of $\mathbf{z}$ and is defined as

$$\bar{\mathbf{z}} = \frac{1}{P}\sum_{p=1}^{P} \mathbf{z}. \tag{D.9}$$

Using equations D.2 and D.8 can rewrite the sum of squared error (D.7) as follows

$$\begin{aligned}\varepsilon &= \frac{1}{2}\sum_{n=M+1}^{I}\sum_{p=1}^{P}\left\{\mathbf{\Phi}^{\top}\left(\mathbf{z}^{p}-\bar{\mathbf{z}}\right)\right\}^{2} \\ &= \frac{1}{2}\sum_{n=M+1}^{I}\mathbf{\Phi}_n^{\top}\Sigma\mathbf{\Phi}_n\end{aligned} \tag{D.10}$$

where $\mathbf{\Sigma}$ is the covariance matrix of the data set $\mathbf{z}$ and is given by

$$\Sigma = \sum_{p=1}^{P}\left(\mathbf{z}^{p}-\bar{\mathbf{z}}\right)\left(\mathbf{z}^{p}-\bar{\mathbf{z}}\right)^{|}. \tag{D.11}$$

It has been shown by many researchers [103, 123], that a solution to equation D.10 occurs with the following constraint,

$$\Sigma\mathbf{\Phi} = \lambda\mathbf{\Phi},$$

where Φ are eigenvectors and λ are eigenvalues of Σ.

Note that with $\mathbf{\Phi}$ being orthonormal the minimum error for a dimensionality reduction is given by

$$\varepsilon = \frac{1}{2}\sum_{n=M+1}^{I}\lambda_n, \tag{D.12}$$

which implies the error is minimized by selecting the $I-M$ smallest eigenvalues and their corresponding eigenvectors, Φ, to discard.

References

[1] Krautkrämer, J. and Krautkrämer, H. *Ultrasounic Testing of Materials.* Springer-Verlag, Berlin, 3rd edition, 1983.

[2] Halmshaw, R. *Non-Destructive Testing*, chapter 6, pages 255–260. Edward Arnold, London, 2nd edition, 1991.

[3] McGonnagle, W. *Nondestructive Testing.* Gordon and Breach, London, 2nd edition, 1982.

[4] Hull, B. and John, V. *Non-destructive Testing.* MacMillan, London, 1988.

[5] Prakash, R. Non-destructive testing of composites. *Composites*, pages 217–224, October 1980.

[6] Halmshaw, R. *Non-Destructive Testing.* Edward Arnold, London, 2nd edition, 1991.

[7] Kalyanasundaram, P., Rajagopalan, C., Subramanian, C. V., Thavasimuthu, M., and Raj, B. Ultrasonic signal analysis for defect characterisation in composite materials. *British Journal of NDT*, 33(5):221–226, May 1991.

[8] Green, R. E. Nondestructive evaluation of materials. *Annual Review of Material Science*, 20:197–217, 1990.

[9] Costanzo, L. Aerospace: Patched up and ready to fly. *Advanced Composites Engineering*, pages 6–7, September 1992.

[10] Scudder, L. P. *Characterisation and Testing of Carbon Fibre Reinforced Polymer Composites using Laser Generated Ultrasound.* PhD thesis, University of Warwick, Department of Engineering, Coventry, October 1994.

[11] Silk, M. G. and Maurice, G. *Ultrasonic Transducers for Non-destructive Testing.* Adam Hilger, Bristol, 1984.

[12] Proctor, T. M. An improved piezoelectric acoustic emission transducer. *Journal for the Acoustical Society of America*, 71:1163–1168, 1982.

[13] Schwartz, M. M. *Composite Materials Handbook.* McGraw-Hill, London, 2nd edition, 1992.

[14] Meyer, R. W. *Handbook of Pultrusion Technology.* Chapman and Hall, London, 1985.

[15] Mayer, R. M. *Design with Reinforced Plastics*, chapter 6, pages 136–138. The Design Council, London, 1993.

[16] Mayer, R. M. *Design with Reinforced Plastics*, chapter 6, pages 139–140. The Design Council, London, 1993.

[17] Stone, D. E. W. and Clarke, B. Non-destructive evaluation of composite structures - an overview. In Matthews, F. L., Buskell, N. C. R., Hodgkinson, J. M., and Morton, J., editors, *Sixth International Conference on Composite Materials*, volume 1, pages 128–159, New York, July 1987. ICCM & ECCM, Elsevier Applied Science.

[18] Hill, S. Rapid nondestructive testing or carbon fibre reinforced plastics. *Materials World, The Journal of the Institute of Metals*, 4(8):450, August 1996.

[19] Balasubramaniam, K., Alluri, S., and Nidumolu, P. Ultrasonic and vibration methods for the characterization of pultruded composites. *Composite Engineering*, 5(12):1433–1451, May 1995.

[20] Fecko, D. L., Steiner, K. V., and Gillespie, J. W. Acousto-ultrasonic inspection of pultruded composites. In Trabocco, R. and Lynch, T., editors, *Advanced*

Materials: Expanding the Horizons, volume 25, pages 940–950. Society for the Advancement of Material and Process Engineering, October 1993.

[21] Dayhoff, J. E. *Neural Network Architectures: An Introduction.* Van Nostrand Reinhold, 1990.

[22] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[23] Mann, J.M., Schmerr, L. W., and Moulder, J. C. Neural network inversion of uniform-field eddy current data. *Materials Evaluation*, pages 34–39, January 1991.

[24] Baxt, W. G. Use of an artificial neural network for data analysis in clinical decision-making: The diagnosis of acute coronary occlusion. *Neural Computation*, 2:480–489, 1990.

[25] Hornik, K. and Stinchcombe, M. Multilayer feedforward networks are universal approximators. In *Artificial Neural Network: Approximation and Learning Theory*, pages 12–28. Blackwell, Oxford, UK, 1992.

[26] Stinchcombe, M. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Artificial Neural Network: Approximation and Learning Theory*, pages 29–40. Blackwell, Oxford, UK, 1992.

[27] Trippi, R. and Turban, E. *Neural Networks in Finance and Investing.* Irwin/Probus Publishing, 1993.

[28] Morgan, N. and Bourlard, H. A. Neural networks for statistical recognition of continuous speech. *Proceedings of the IEEE*, 83(5):742–770, 1995.

[29] Calabro, C., Ferragina, P., and Granieri, M. N. Recognition of hand-written rotated digits by neural networks. *Machine Vision and Applications*, 8(5):351–357, 1995.

[30] Baxt, W. G. Application of artificial neural networks to clinical medicine. *The Lancet*, 346(8983):1135–1138, October 1995.

[31] Guglielmi, N., Guerrieri, R., and Baccarani, G. Highly constrained neural networks for industrial quality-control. *IEEE Transactions on Neural Networks*, 7(1):206–213, 1996.

[32] Smith, P. R., Green, D. A., and Naimimohasses, R. Some investigations on neural processing of scattered-light in water-quality assessment. *IEE Proceedings-Vision and Signal Processing*, 141(4):261–266, 1994.

[33] Hakulinen, A. and Hakkarainen, J. A neural network approach to quality control of padlock manufacturing. *Pattern Recognition Letters*, 17(4):357–362, 1996.

[34] Hopgood, A. A., Hallam, N. J., and Woodcock, N. An on-line knowledge-based system for interpreting ultrasonic data. In *Colloquium on Expert Systems for NDT*, February 1989.

[35] O'Rorke, P. and Morris, S. Abductive signal interpretation for nondestructive evaluation. In Biswas, G., editor, *Applications of Artificial Intelligence X: Knowledge-Based Systems*, volume 1707, pages 68–75, Florida, April 1992. Society for Optical Engineering, SPIE.

[36] Windsor, C. G. The classification of defects from ultrasonic measurements. In *Colloquium on Expert Systems for NDT*, February 1989.

[37] McNab, A. and Dunlop, I. A review of artifical intelligence applied to ultrasonic defect evaluation. *INSIGHT Non-Destructive Testing and Condition Monitoring*, 37(1):11–16, January 1995.

[38] Windsor, C. G. Can we train a computer to be a skilled inspector. *INSIGHT Non-Destructive Testing and Condition Monitoring*, 37(1):36–49, January 1995.

[39] Udpa, L. and Udpa, S. S. Neural networks for the classification of nondestructive signals. *IEE Proceedings-F Radar and Signal Processing*, 138(1):41–45, February 1991.

[40] Udpa, L., Berry, D., and Udpa, S. S. Ultrasonic signal classification using artificial neural networks. *Ultrasonics International*, 1991.

[41] Damarla, T. R., Karpur, P., and Bhagat, P. K. A self-learning neural net for ultrasonic signal analysis. *Ultrasonics*, 30(5):317–324, 1992.

[42] Challis, R. E. and Bork, U. Artificial neural network preprocessing of lamb wave data for adhesive bond characterization. In *Proceedings of the Ultrasonics International Conference*, pages 775–778, London, July 1993. University of Keele, Butterworth-Heinemann.

[43] Bork, U. and Challis, R. E. Nondestructive evaluation of the adhesive fillet size in a t-peel joint using ultrasonic lamb waves and a linear-network for data discrimination. *Measurement Science and Technology*, 6(1):72–84, 1995.

[44] Sachse, W., Grabec, I., and Sribar, R. Intelligent processing of ultrasonic signals for quantitative materials testing. In *Ultrasonics Symposium*, pages 767–775, 1991.

[45] Bull, D. R. and Smith, P. D. Neural network based object classification using an ultrasonic array. In *Sensor Update: New Developments in Signal Processing*, pages 116–127. University of Southampton, September 1993.

[46] Shahani, K., Udpa, L., and Udpa, S. S. Neural networks for classification of ultrasonic NDE signals. *1st Proceedings of IEE ICANN*, pages 407–412, 1990.

[47] Baker, A. R. and Windsor, C. G. The classification of defects from ultrasonic data using neural networks: The hopfield method. *NDT International*, 22(2):97–105, 1989.

[48] Thomsen, J. J. and Lund, K. Quality control of composite materials by neural network analysis of ultrasonic power spectra. *Materials Evaluation*, pages 594–600, May 1991.

[49] Scudder, H. J. Introduction to computer aided tomography. *Proceedings of the IEEE*, 66(6):628–637, June 1978.

[50] Jansen, D. P. *Acoustic Tomographic Imaging of Solid Media.* PhD thesis, Queen's University, Kingston, Ontario, Canada, March 1992.

[51] Tsao, M. C. Industrial ultrasonic tomography - principle, practice, and limitation. *Materials Evaluation*, 41:1248–1254, October 1983.

[52] Mueller, R. K., Kaveh, M., and Wade, G. Reconstructive tomography and applications to ultrasonics. *Proceedings of the IEEE*, 67(4):567–587, April 1979.

[53] Jones, H. W. Recent activity in ultrasonic tomography. *Ultrasonics*, 31(5):353–360, 1993.

[54] Wells, P. N. T. The present status of ultrasonic imaging in medicine. *Ultrasonics*, 31(5):345–352, 1993.

[55] Sponheim, N., Gelius, L. J., Johansen, I., and Stamnes, J. J. Ultrasonic tomography of biological tissue. *Ultrasonic Imaging*, 16(1):19–32, 1994.

[56] Hamamoto, K., Andreas, B., Shina, T., and Ito, M. Basic investigation of reflection mode ultrasonic-attenuation tomography. *Japanese Journal of Applied Physics Part 1*, 34(5B):2812–2816, 1995.

[57] Chapman, C. H. The radon transform and seismic tomography. In Nolet, G., editor, *Seismic Tomography*, pages 25–48. D. Reidel Publishing Co., Dordrect, Holland, 1987.

[58] Hope, V. S. The application of seismic velocity tomography in geotechnical investigations. *Proceedings of the Institution of Civil Engineers-Geotechnical Engineering*, 113(4):215–225, 1995.

[59] Watanabe, T. and Sassa, K. Seismic attenuation tomography and its applications to rock mass evaluation. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, 33(5):467–477, 1996.

[60] Conrath, B. C., Daft, C. M. W., and O'Brien, W. D. Applications of neural networks to ultrasound tomography. In *Ultrasonics Symposium*, pages 1007–1010, 1989.

[61] McCulloch, W. S. and Pitts, W. H. A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[62] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[63] Rosenblatt, F. *Principals of Neurodynamics.* Spartan, New York, 1962.

[64] Kohonen, T. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 3 edition, 1989.

[65] Rumelhart, D. E., McClelland, J. L., and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, 1986.

[66] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1, chapter 8, pages 318–362. MIT Press, Cambridge, 1986.

[67] Widrow, B. and Hoff, M. E. Adaptive switching circuits. In *1960 IRE Western Electric Show and Convention Record*, pages 96–104, August 1960.

[68] Widrow, B. Generalization and information storage in networks of Adaline 'neurons'. In Jovitz, M. C., Jacobi, G. T., and Goldstein, G., editors, *Self-organizing Systems*, pages 435–461. Spartan Books, Washington, D.C., 1962.

[69] Hebb, D. O. *The Organisation of Behaviour.* Wiley, New York, 1949.

[70] Zurada, J. M. *Introduction to Artificial Neural Systems.* West, New York, 1992.

[71] Fahlman, S. E. Faster–learning variations on back–propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, September 1989.

[72] Hinton, G. E. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[73] Kohonen, T. Self-organising maps - optimisation approaches. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 981–990, Finland, 1991.

[74] Moody, J. and Darken, C. J. Fast learning in networks of locally-tuned processing units. *Neural Computation*, pages 281–294, June 1989.

[75] Miyata, Y. *A User's Guide to PlaNet.* Computer Science Department, University of Colorado, Boulder, January 1991.

[76] Regier, T. Quickprop. Translation of Scott Fahlman's quickprop program from Common Lisp into C, September 1988.

[77] Goodman, P., Rosen, D., and Plummer, A. *NevProp.* University of Nevada Center for Biomedical Modeling Research, Washoe Medical Center, NV, July 1993.

[78] NeuralWare Inc. *NeuralWorks Professional II/Plus.*

[79] Lari-Najafi, H., Nasiruddin, M., and Samad, T. Effect of initial weights on back–propagation and its variations. *IEEE Transactions on Systems Man and Cybernetics*, pages 218–219, November 1989.

[80] Haykin, S. *Neural Networks: A Comprehensive Foundation.* Macmillan College Publishing, New York, 1994.

[81] Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., and Hopfield, J. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

[82] Widrow, B. and Lehr, M. A. 30 years of adaptive neural networks: Perceptron madaline and backpropagation. *Proceedings of the IEEE*, 78(9):82–108, September 1990.

[83] Cover, T. M. Geometrical and statistical properties of linear threshold devices. Technical report 6107–1, Stanford Electronic Laboratories, Stanford, CA, May 1964.

[84] Cover, T. M. Capacity problems for linear machines. In Kanal, L. N., editor, *Pattern Recognition*, part 3, pages 283–289. Thompson Book Co., Washington DC, 1968.

[85] Baum, E. B. On the capabilities of multilayer perceptrons. *Journal of Complexity*, pages 193–215, March 1988.

[86] Kolmogorov, A. N. On the representation of continuous functions of many variables by superposition of continous functions of one variable and addition. *Dokl. Akad. Nauk USSR*, pages 953–956, 1957.

[87] Lorentz, G. G. The 13th problem of Hillbert. In *Proceedings of the Symposia in Pure Mathematics*, volume 28. American Mathematical Society, 1976.

[88] Sprecher, D. A. On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115:340–355, March 1965.

[89] Hecht-Nielsen, R. Kolmogorov's mapping neural network existence theorem. In *1st IEEE International Conference on Neural Networks*, volume 3, pages 11–14, San Diego, June 1987.

[90] Villiers, J. and Barnard, E. Backpropagation neural nets with one and two hidden layers. *IEEE Transactions on Neural Networks*, 4(1):136–141, January 1992.

[91] LeCun, Y. *Generalisation and Network Design Strategies.* Elsevier Science Publishers, 1989.

[92] Minnix, J. I., McVey, E.S., and Inigo, R.M. A multi-layered self-organising ANN for invariant pattern recognition. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):162–167, April 1992.

[93] Girosi, F. and Poggio, T. Representation properties of networks: Kolmogorov's theorem is irrelevant. *Microcognition: philosophy cognitive science and parallel distributed processing.*, pages 83–105, July 1989.

[94] DeRouin, E. and Brown, J. Neural network training on unequally represented classes. In *Proceedings of the ANN in Engineering Conference*, pages 135–140. ASME Press, November 1991.

[95] Anand, R., Mehrotra, K. G., Mohan, C. K., and Ranka, S. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, November 1993.

[96] Weiss, S. M. and Kulikowski, C. A. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Networks, Machine Learning and Expert Systems*, chapter 2, pages 17–49. Morgan Kaufmann, San Mateo, California, 1991.

[97] Eberhart, R. C., Dobbins, R. W., and Hutton, L. V. *Neural Network PC Tools: A Practical Guide*, chapter 7, pages 161–176. Academic Press, New York, 1990.

[98] MathWorks Inc. *Matlab*, version 4.2c, Nov 1994.

[99] Viktorov, I. A. *Rayleigh and Lamb Waves: Physical Theory and Applications.* Plenum Press, New York, 1967.

[100] Li, H. U. and Negishi, K. Visualization of lamb mode patterns in a glass plate. *Ultrasonics*, 32(4):243–248, 1994.

[101] Bishop, N. W. M. and Sherratt, F. Fatigue life prediction from power spectral density data. *Environmental Engineering*, 2(2), June 1989.

[102] Pearson, K. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[103] Jolliffe, I. T. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[104] Thodberg, H. H. A review of bayesian neural networks with an application to near infrared spectroscopy. *IEEE Transactions on Neural Networks*, 7(1):57–70, January 1996.

[105] Montgomery, D. C. and Peck, E. A. *Introduction to Linear Regression Analysis.* Wiley, New York, 1982.

[106] Draper, N. R. and Smith, H. *Applied Regression Analysis.* Wiley, New York, 2nd edition, 1981.

[107] Sarle, W. S. Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC, USA, April 1994. SAS Institute.

[108] Lewitt, R. M. Reconstructional algorithms: Transform methods. *Proceedings of the IEEE*, 71:409–419, 1983.

[109] Kline, R. A. and Wang, Y. Q. A technique for ultrasonic tomography in anisotropic media. *Journal of the Acoustic Society of America*, 91(2):878–884, February 1992.

[110] Censor, Y. Finite series-expansion reconstruction methods. *Proceedings of the IEEE*, 71:103–112, 1983.

[111] Duncan, P. J., Gaydecki, P. A., and Burdekin, F. M. C-scan image enhancement by digitial frequency domain block filtering. *INSIGHT Non-Destructive Testing and Condition Monitoring*, 37(1):43–49, January 1995.

[112] Jain, A. K. and Dubuisson, M. Segmentation of X-ray and C-scan images of fibre reinforced composite materials. *Pattern Recognition*, 25(3):257–270, 1992.

[113] Wooh, S. C. and Daniel, I. M. Enhancement techniques for ultrasonic nondestructive evaluation of composite materials. *Journal of Engineering Materials and Technology*, 112:175–182, April 1990.

[114] Wang, Y. M. and Frieden, B. R. Minimum entropy-neural network approach to turbulent-image reconstruction. *Applied Optics*, 34(26):5938–3944, 1995.

[115] Wang, W. L. and Whitehouse, D. J. Application of neural network to the reconstruction of scanning probe microscope images distorted by finite-size tips. *Nanotechnology*, 6(2):45–51, 1995.

[116] Daaland, A. The FSM method for monitoring process pipelines offshore. *INSIGHT Non-Destructive Testing and Condition Monitoring*, 38(6):434–439, June 1996.

[117] Strommen, R. D., Horn, H., and Wold, K. R. FSM-a unique method for monitoring corrosion of steel piping and vessels. *Materials Performance*, 32(3):50–54, March 1993.

[118] Oppermann, W. and Keller, H. P. An improved potential drop method for measuring and monitoring defects in metallic structures. *Nuclear Engineering and Design*, 144(1):171–175, 1993.

[119] Frise, P. R. and Sahney, R. Selection of a potential drop crack measurement system for zirconium alloy specimens. *INSIGHT Non-Destructive Testing and Condition Monitoring*, 38(2):96–101, February 1996.

[120] Wojcik, A. G. Potential drop techniques for crack characterization. *Materials World*, 3(8):379–381, 1995.

[121] CorrOcean Ltd. *FSM-The Electric Fingerprint.* Data sheet no. 190.

[122] Ghajarieh, R., Saka, M., Abe, H., Komura, I., and Sakamoto, H. Simplified NDE of multiple cracks by means of the potential drop technique. In *NDT & E International*, volume 28, pages 23–28, 1995.

[123] Bishop, C. M. *Neural Networks for Pattern Recognition.* Oxford University Press, 1996.

Bibliography

Guide to examinations for higher degrees by research. Warwick Graduate School, 1995.

Bentley, J. L. *Programming Perls.* Addison-Wesley, Wokingham, UK, 1986.

Goossens, M., Mittelbach, F., and Samarin, A. *The LaTeX Companion.* Addison-Wesley, Wokingham, 1994.

Holub, A. I. *Enough rope to shoot yourself in the foot, rules for C and C++ programming.* McGraw-Hill, London, 1995.

Kosko, B. *Fuzzy Thinking, the new Science of Fuzzy Logic.* Flamingo, London, 1994.

Lamport, L. *LaTeX, a document preparation system, user's guide and reference manual.* Addison-Wesley, Wokingham, UK, 2nd edition, 1994.

Masters, T. *Signal and Image Processing with Neural Networks.* J. Wiley & Sons, New York, 1994.

Pao, Y. H. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, Wokingham, UK, 1989.

Rich, E. and Knight, K. *Artificial Intelligence.* McGraw-Hill, London, 2nd edition, 1991.

Ripley, B. D. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

Sokolowski, R. Natural and artificial intelligence. In Graubard, S. R., editor, *The Artificial Intelligence Debate: False Starts, Real Foundations*, chapter 3, pages 45–64. MIT Press, Cambridge, 1988.

Stroustrup, B. *The C++ Programming Language.* Addison-Wesley, Wokingham, UK, 2nd edition, April 1995.

Watson, M. *C++ power paradigms.* McGraw-Hill, New York, 1994.